US006285658B1

(12) United States Patent
Packer

(10) Patent No.: US 6,285,658 B1
(45) Date of Patent: *Sep. 4, 2001

(54) SYSTEM FOR MANAGING FLOW BANDWIDTH UTILIZATION AT NETWORK, TRANSPORT AND APPLICATION LAYERS IN STORE AND FORWARD NETWORK

(75) Inventor: Robert L. Packer, Los Gatos, CA (US)

(73) Assignee: Packeteer, Inc., Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: 09/479,356

(22) Filed: Jan. 7, 2000

Related U.S. Application Data

(63) Continuation of application No. 08/977,642, filed on Nov. 24, 1997, now Pat. No. 6,046,980.
(60) Provisional application No. 60/032,485, filed on Dec. 9, 1996.

(51) **Int. Cl.**^{7} **G01R 31/08**
(52) **U.S. Cl.** **370/230**; 370/229
(58) **Field of Search** 370/464, 465, 370/466, 468, 229, 230, 231, 233, 234, 232, 236, 471, 445, 241, 252, 235; 707/229, 250

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,870,641		9/1989	Pattavina	370/60
5,315,586		5/1994	Charvillat	370/60
5,347,511		9/1994	Gun	370/54
5,442,630		8/1995	Gagliadi et al.	370/85
5,506,834		4/1996	Sekihata et al.	370/17
5,530,852		6/1996	Meske, Jr. et al.	395/600
5,583,857	*	12/1996	Soumiya	370/234
5,694,548		12/1997	Baugher	370/231
5,701,465		12/1997	Baugher	370/468
5,732,219		3/1998	Blumer et al.	395/200.57
5,748,629	*	5/1998	Caldara	370/468
5,764,645	*	6/1998	Bernet	370/466

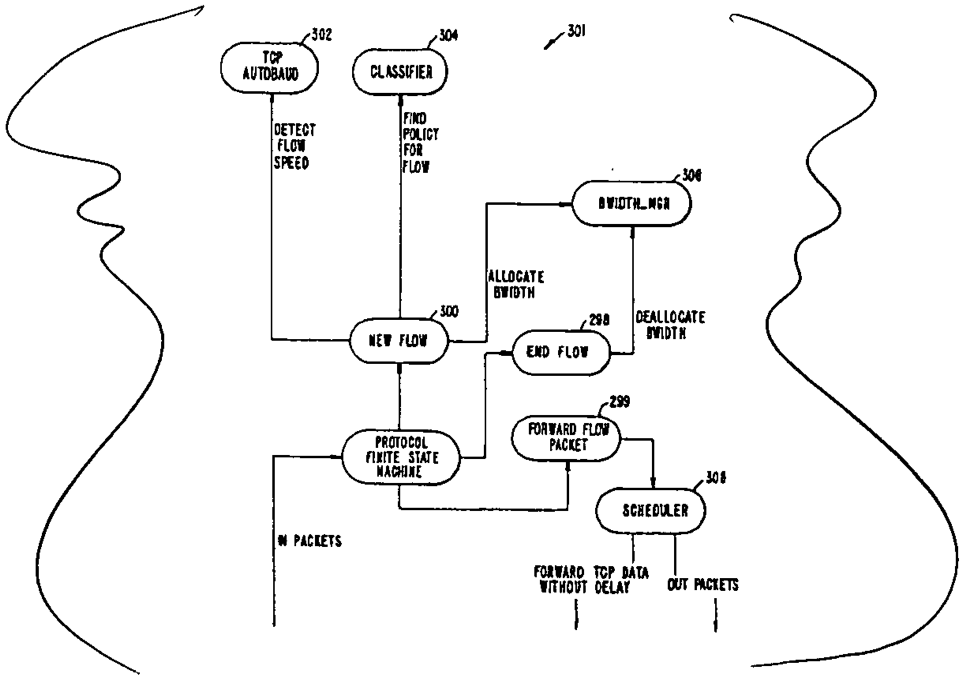
* cited by examiner

Primary Examiner—Douglas Olms
Assistant Examiner—Ricardo M. Pizarro
(74) *Attorney, Agent, or Firm*—Townsend and Townsend and Crew LLP; Kenneth R. Allen

(57) **ABSTRACT**

In a packet communication environment, a method is provided for classifying packet network flows for use in determining a policy, or rule of assignment of a service level, and enforcing that policy by direct rate control. The method comprises applying individual instances of traffic objects, i.e., packet network flows to a classification model based on selectable information obtained from a plurality of layers of a multi-layered communication protocol, then mapping the flow to the defined traffic classes, which are arbitrarily assignable by an offline manager which creates the classification. It is useful to note that the classification need not be a complete enumeration of the possible traffic.

17 Claims, 20 Drawing Sheets



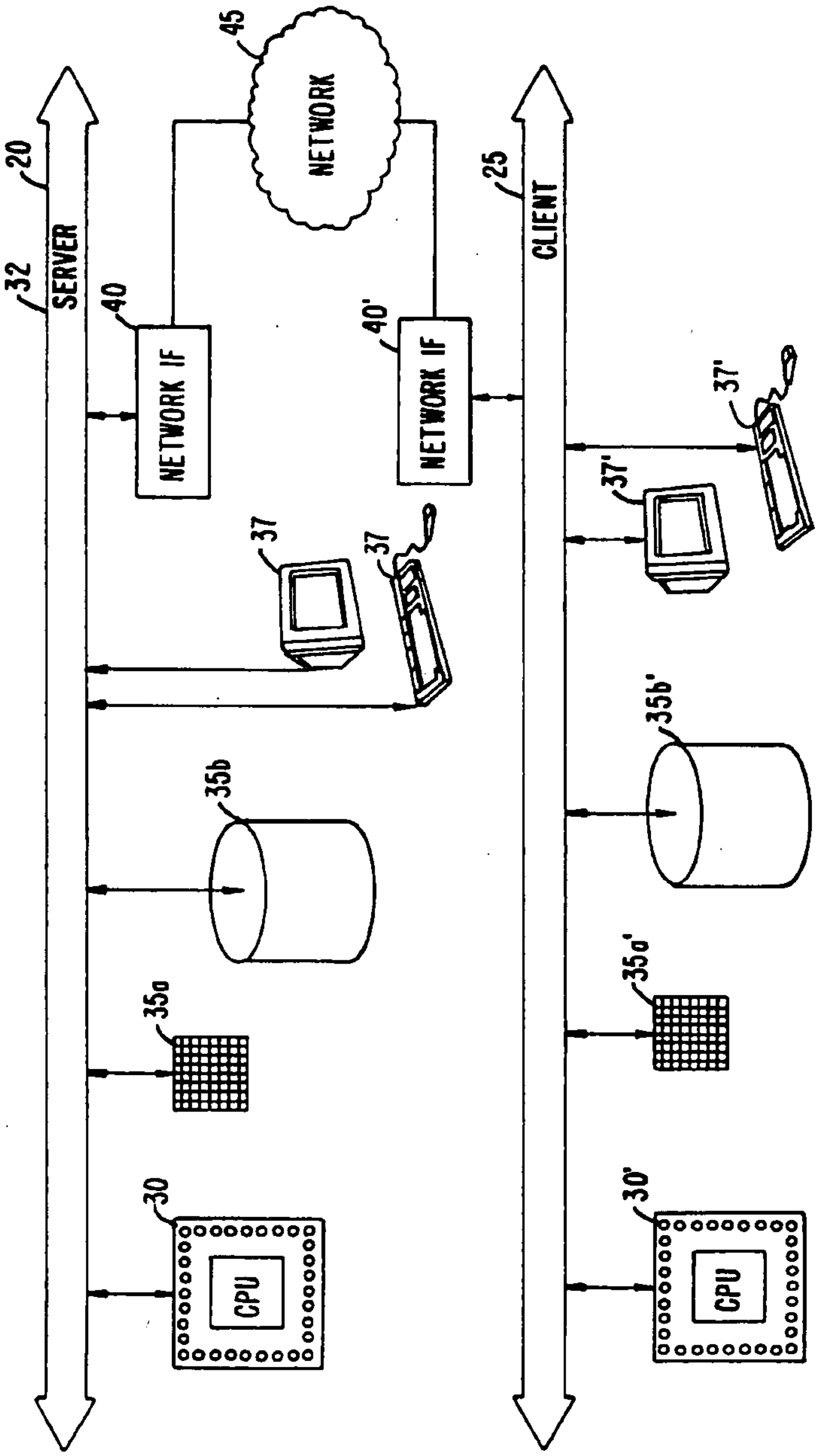


FIG. 1A.
(PRIOR ART)

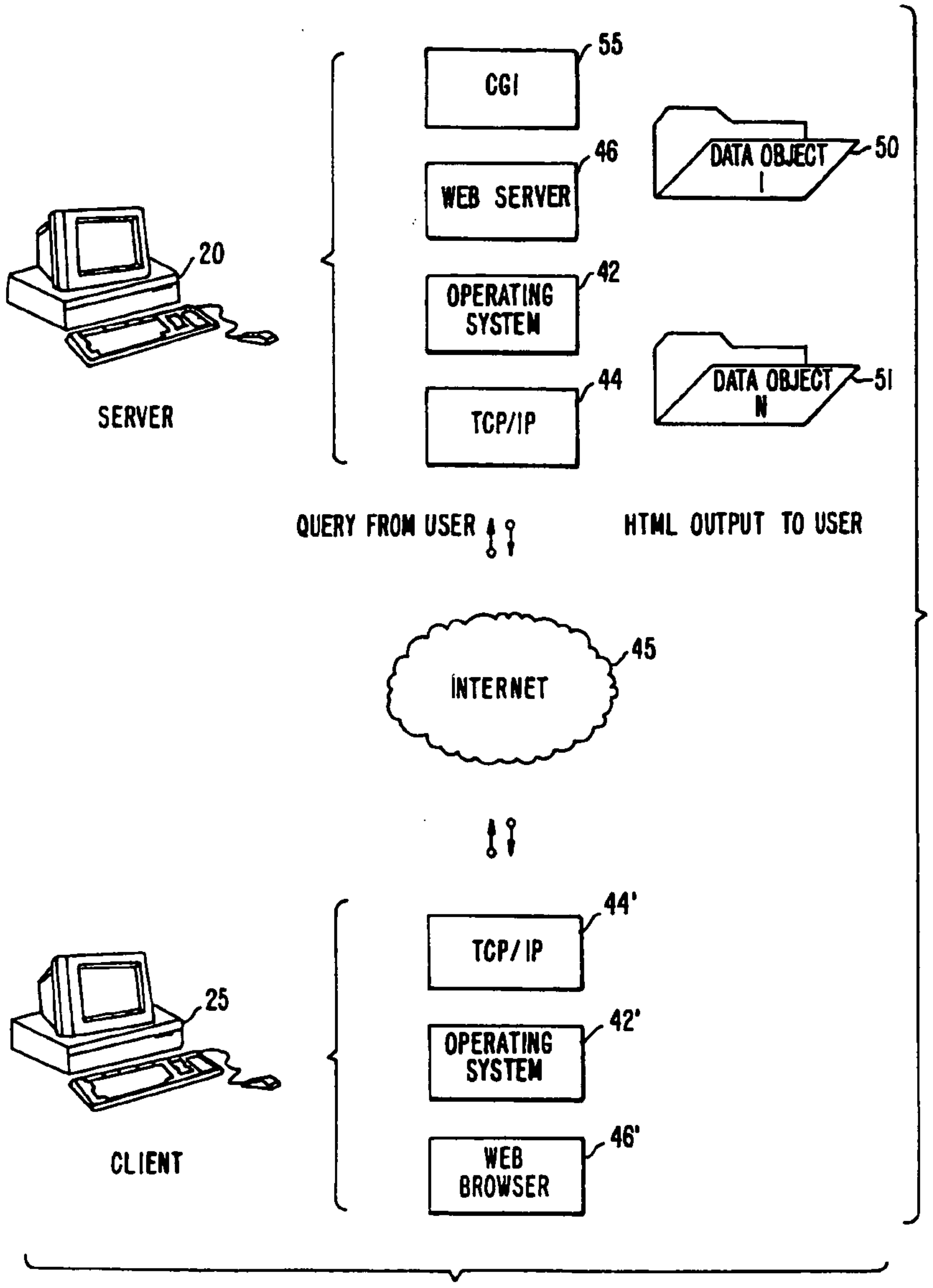

55
CGI
46
WEB SERVER
42
OPERATING SYSTEM
44
TCP/IP
DATA OBJECT 1
50
DATA OBJECT N
51
20
SERVER
QUERY FROM USER
HTML OUTPUT TO USER
INTERNET
45
44'
TCP/IP
25
42'
OPERATING SYSTEM
46'
WEB BROWSER
CLIENT

FIG. 1B.

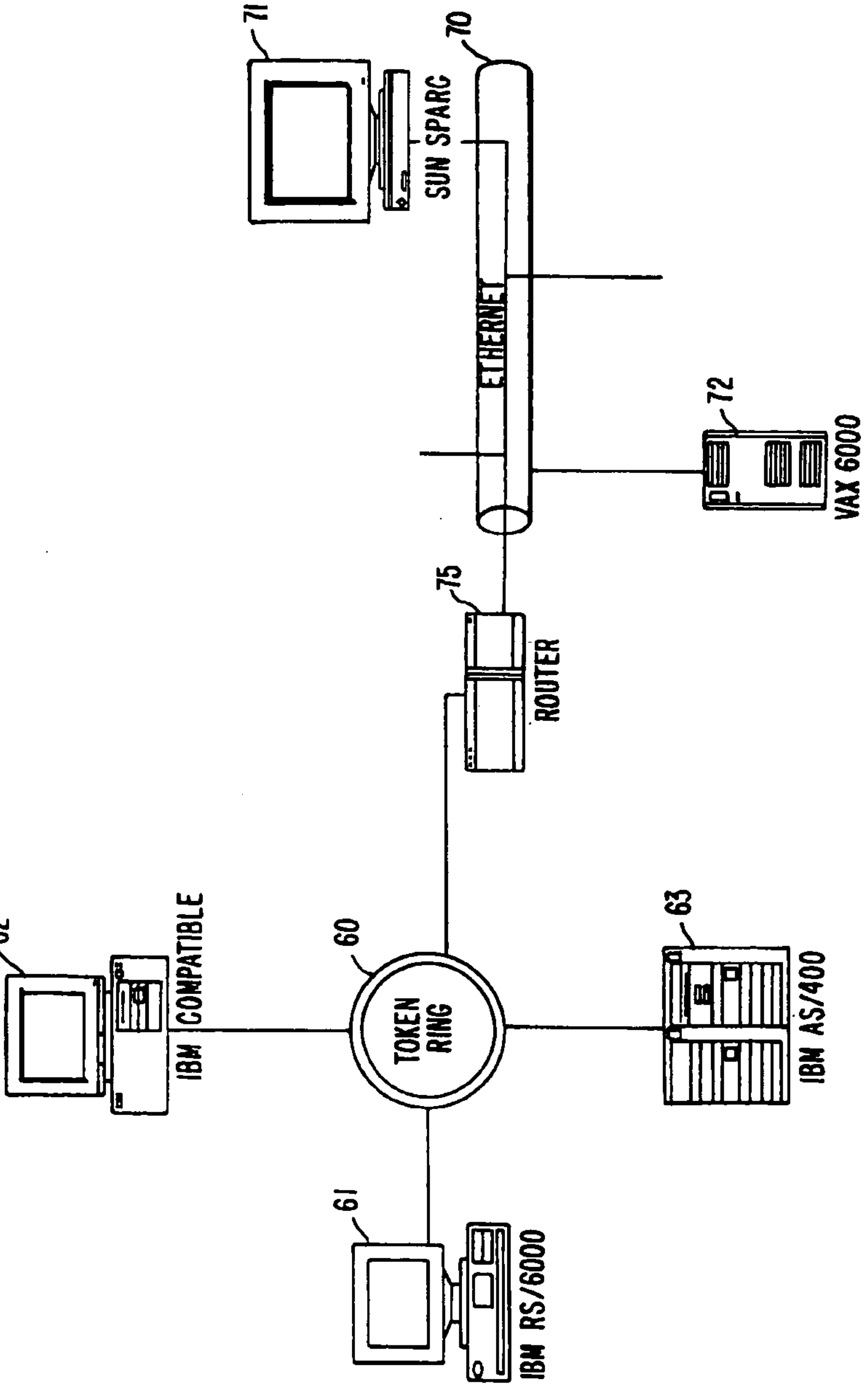
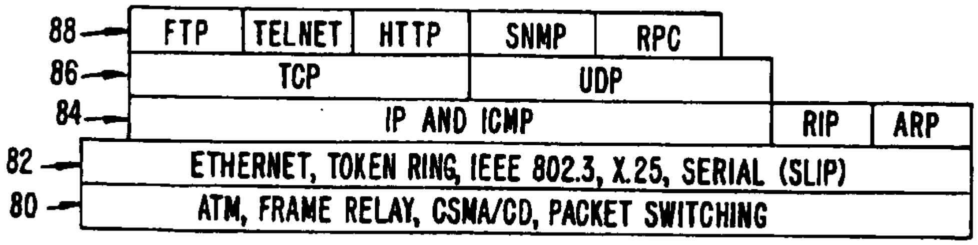


FIG. 1C. (PRIOR ART)


88
86
84
82
80
FTP
TELNET
HTTP
SNMP
RPC
TCP
UDP
IP AND ICMP
RIP
ARP
ETHERNET, TOKEN RING, IEEE 802.3, X.25, SERIAL (SLIP)
ATM, FRAME RELAY, CSMA/CD, PACKET SWITCHING

LEGEND

88	SESSION/APPLICATION LAYER
86	TRANSPORT LAYER
84	NETWORK LAYER
82	DATA LINK LAYER
80	PHYSICAL LAYER

FIG. 1D.

(PRIOR ART)

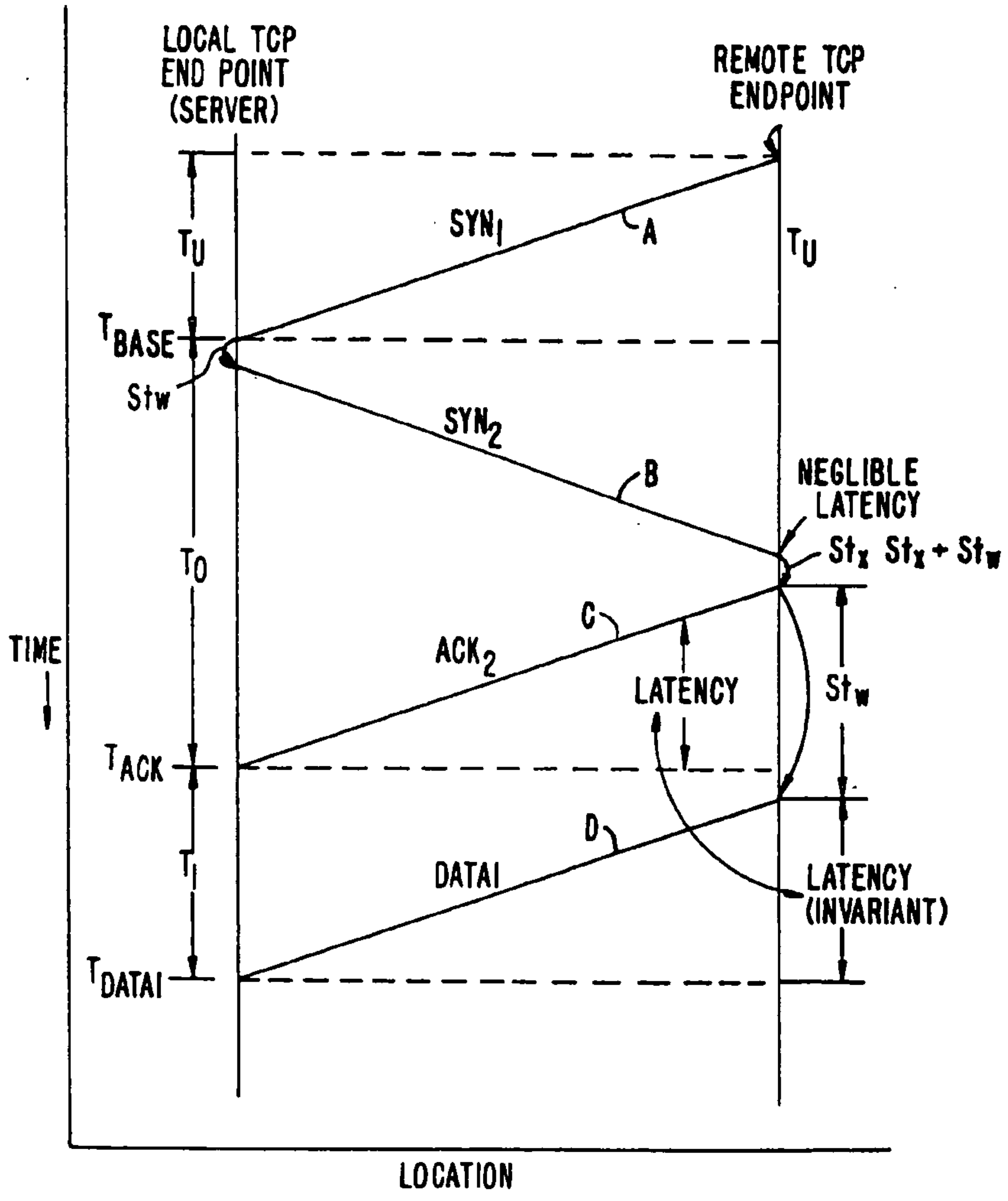


FIG. 1E.

(PRIOR ART)

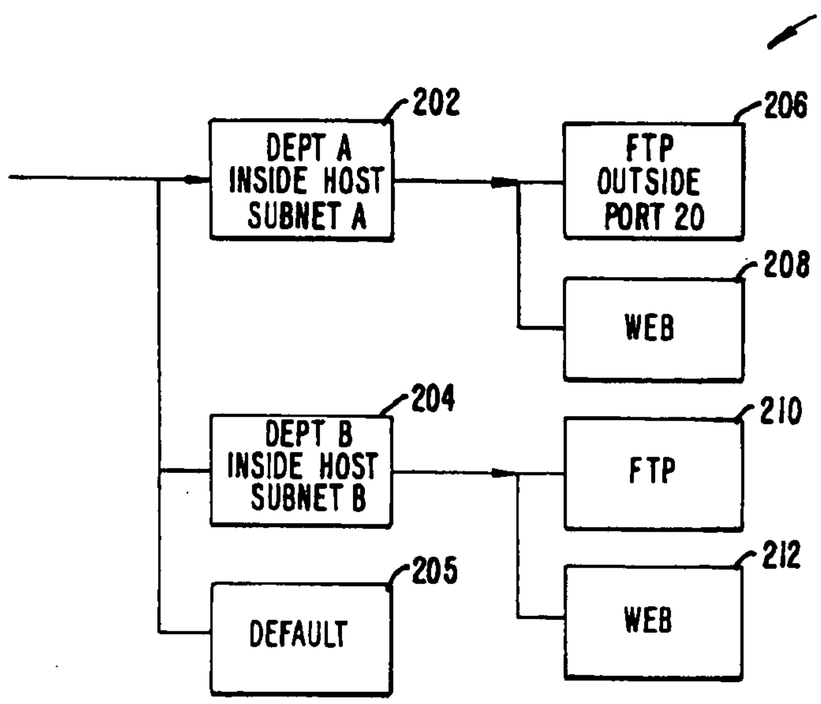

201
202
DEPT A INSIDE HOST SUBNET A
206
FTP OUTSIDE PORT 20
208
WEB
204
DEPT B INSIDE HOST SUBNET B
210
FTP
205
DEFAULT
212
WEB

FIG. 2A.

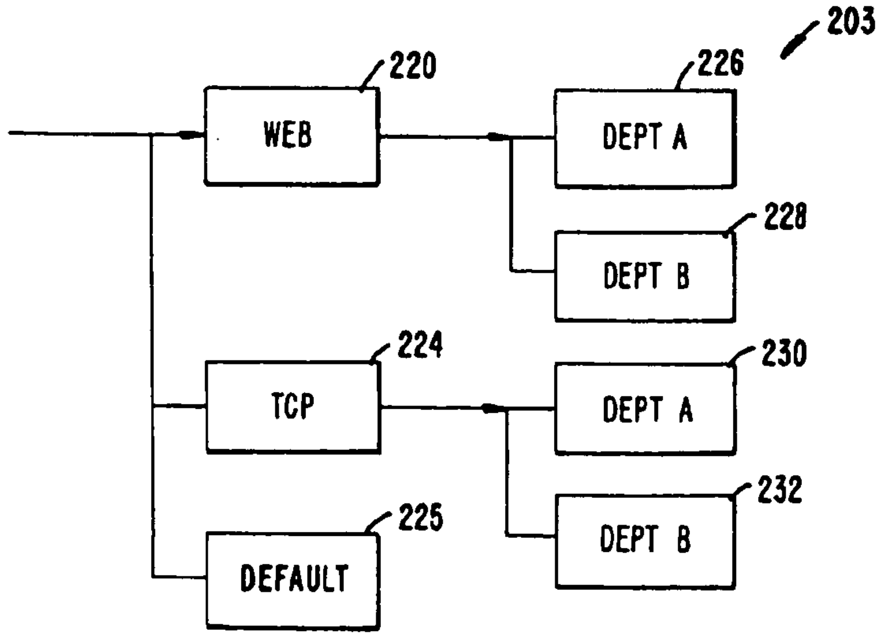

203
220
WEB
226
DEPT A
228
DEPT B
224
TCP
230
DEPT A
225
DEFAULT
232
DEPT B

FIG. 2B.

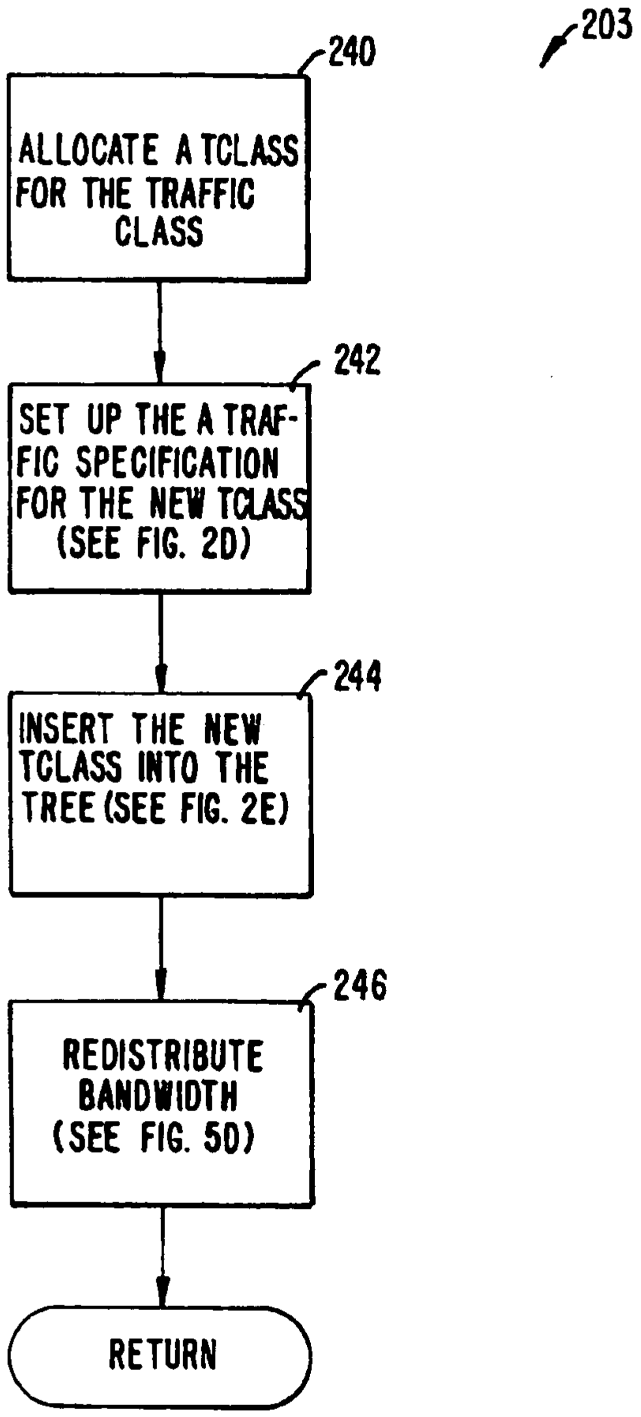

203
240
ALLOCATE A TCLASS FOR THE TRAFFIC CLASS
242
SET UP THE A TRAFFIC SPECIFICATION FOR THE NEW TCLASS (SEE FIG. 2D)
244
INSERT THE NEW TCLASS INTO THE TREE (SEE FIG. 2E)
246
REDISTRIBUTE BANDWIDTH (SEE FIG. 5D)
RETURN

FIG. 2C.

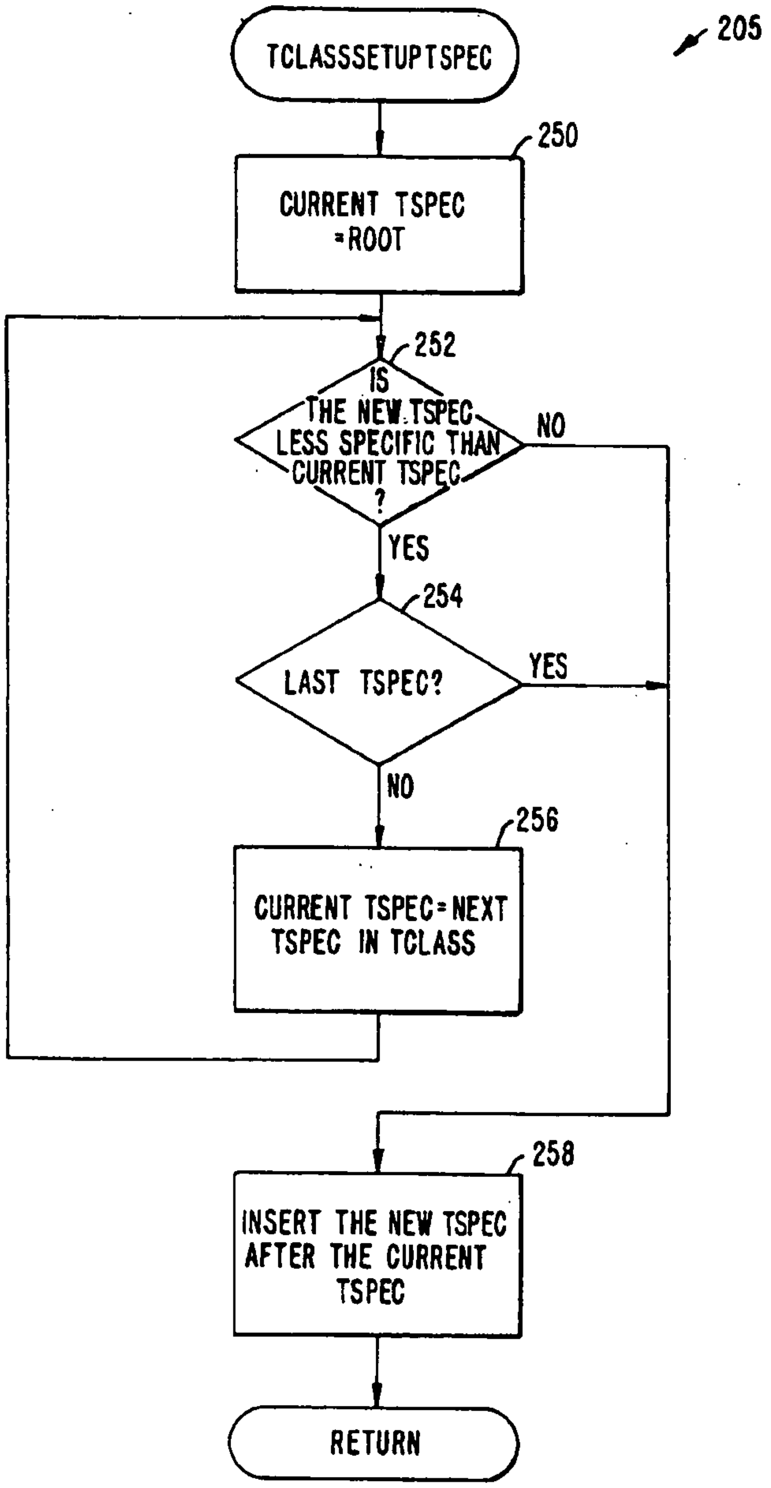
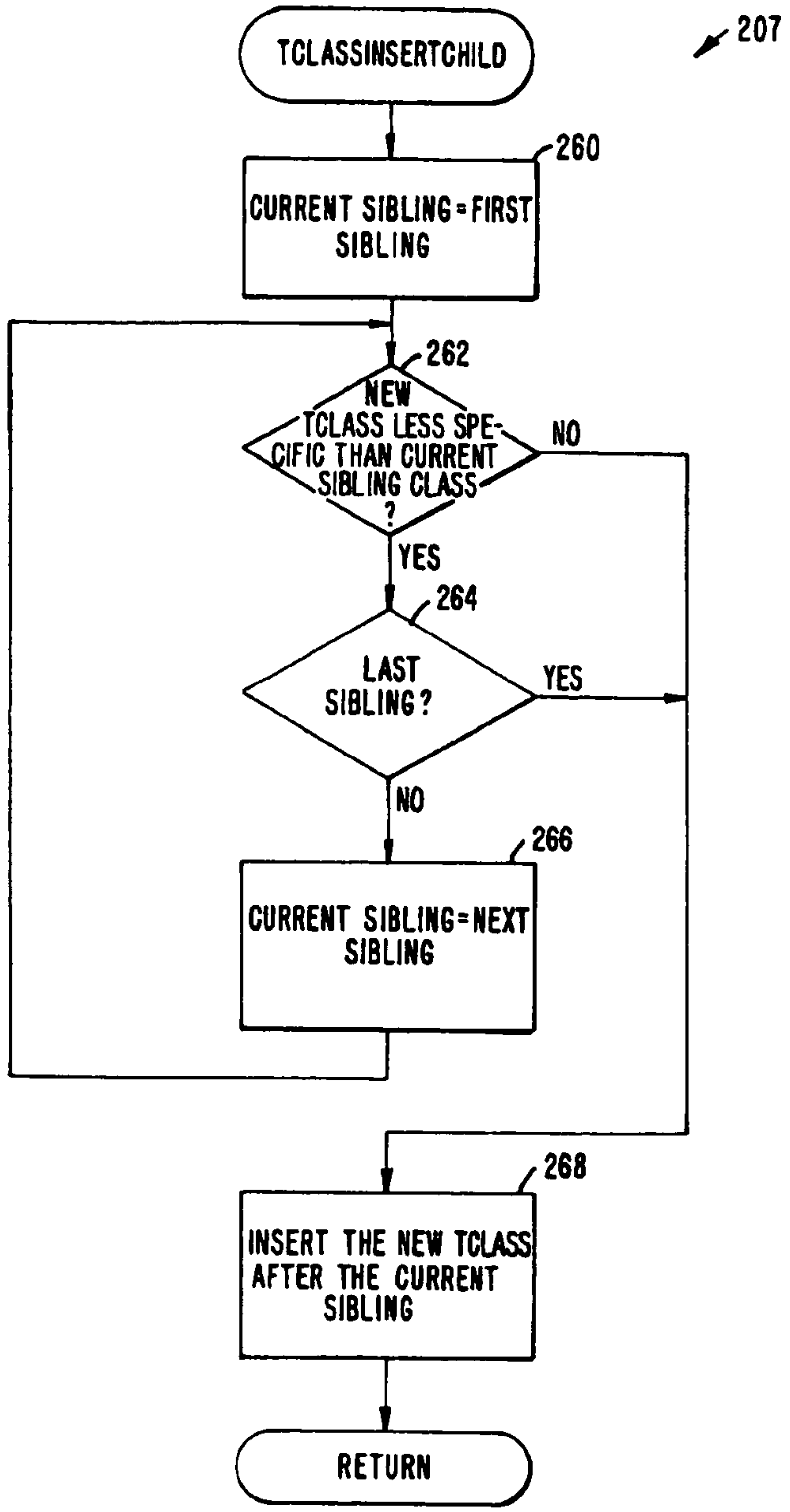

205
TCLASSSETUPTSPEC
250
CURRENT TSPEC =ROOT
252
IS THE NEW TSPEC LESS SPECIFIC THAN CURRENT TSPEC ?
NO
YES
254
LAST TSPEC?
YES
NO
256
CURRENT TSPEC=NEXT TSPEC IN TCLASS
258
INSERT THE NEW TSPEC AFTER THE CURRENT TSPEC
RETURN

FIG. 2D.



207
TCLASSINSERTCHILD
260
CURRENT SIBLING=FIRST SIBLING
262
NEW TCLASS LESS SPECIFIC THAN CURRENT SIBLING CLASS ?
NO
YES
264
LAST SIBLING?
YES
NO
266
CURRENT SIBLING=NEXT SIBLING
268
INSERT THE NEW TCLASS AFTER THE CURRENT SIBLING
RETURN

FIG. 2E.

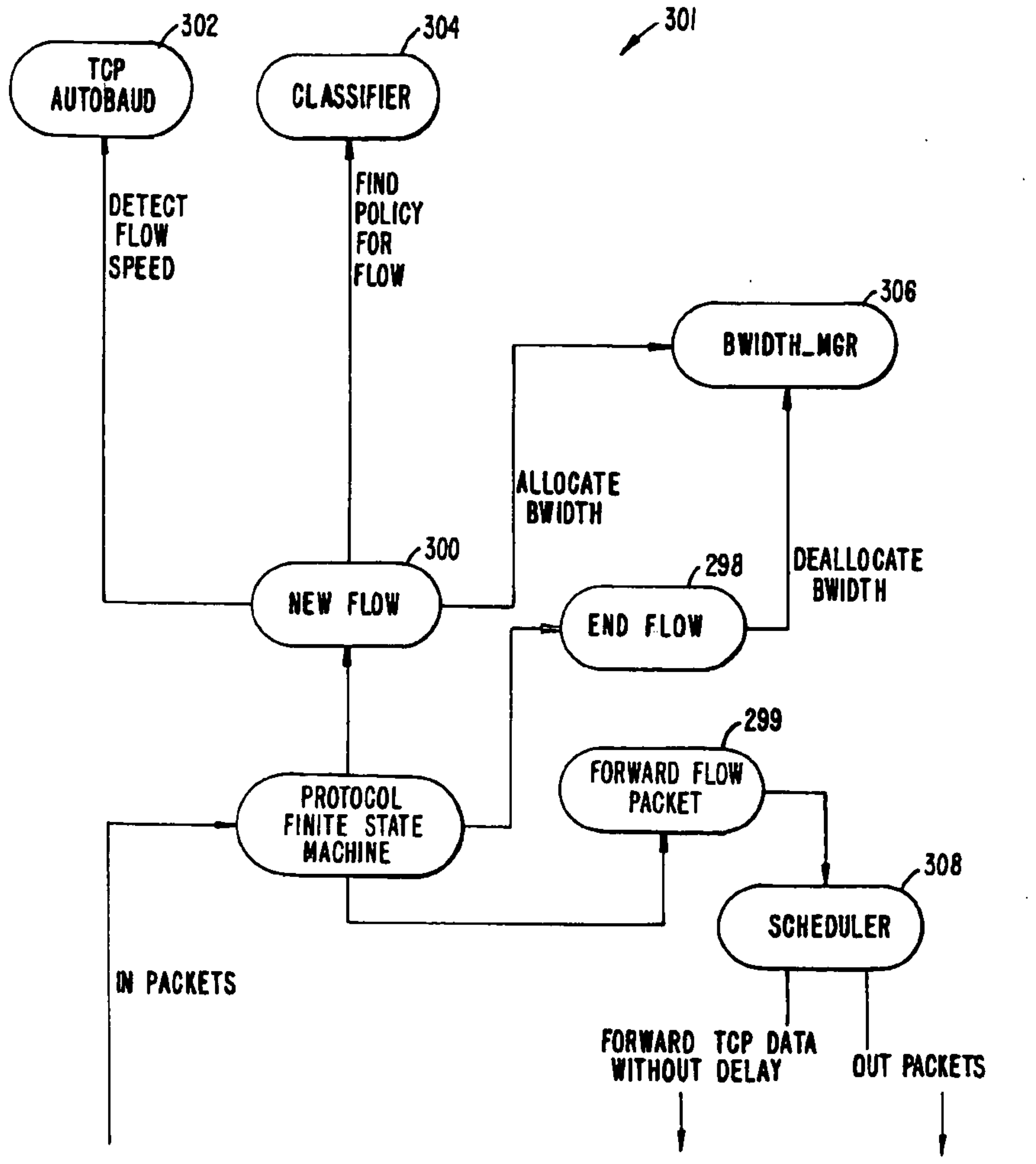
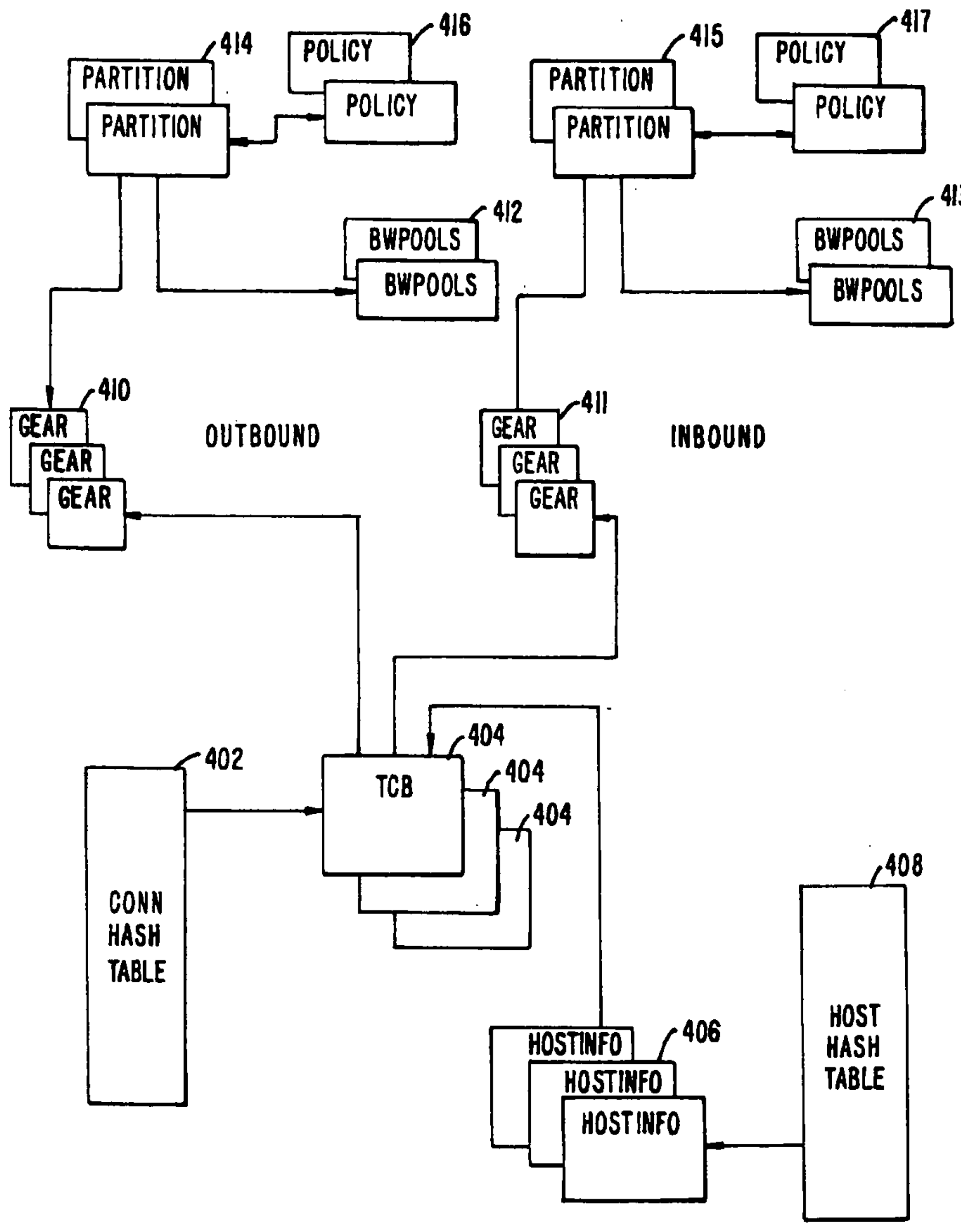

302
TCP AUTOBAUD
304
CLASSIFIER
301
DETECT FLOW SPEED
FIND POLICY FOR FLOW
306
BWIDTH_MGR
ALLOCATE BWIDTH
300
NEW FLOW
298
END FLOW
DEALLOCATE BWIDTH
299
FORWARD FLOW PACKET
PROTOCOL FINITE STATE MACHINE
308
SCHEDULER
IN PACKETS
FORWARD TCP DATA WITHOUT DELAY
OUT PACKETS

FIG. 3.



PARTITION
PARTITION
414
POLICY
POLICY
416
PARTITION
PARTITION
415
POLICY
POLICY
417
BWPOOLS
BWPOOLS
412
BWPOOLS
BWPOOLS
413
GEAR
GEAR
GEAR
410
OUTBOUND
GEAR
GEAR
GEAR
411
INBOUND
CONN HASH TABLE
402
TCB
404
404
404
HOSTINFO
HOSTINFO
HOSTINFO
406
HOST HASH TABLE
408

FIG. 4A.

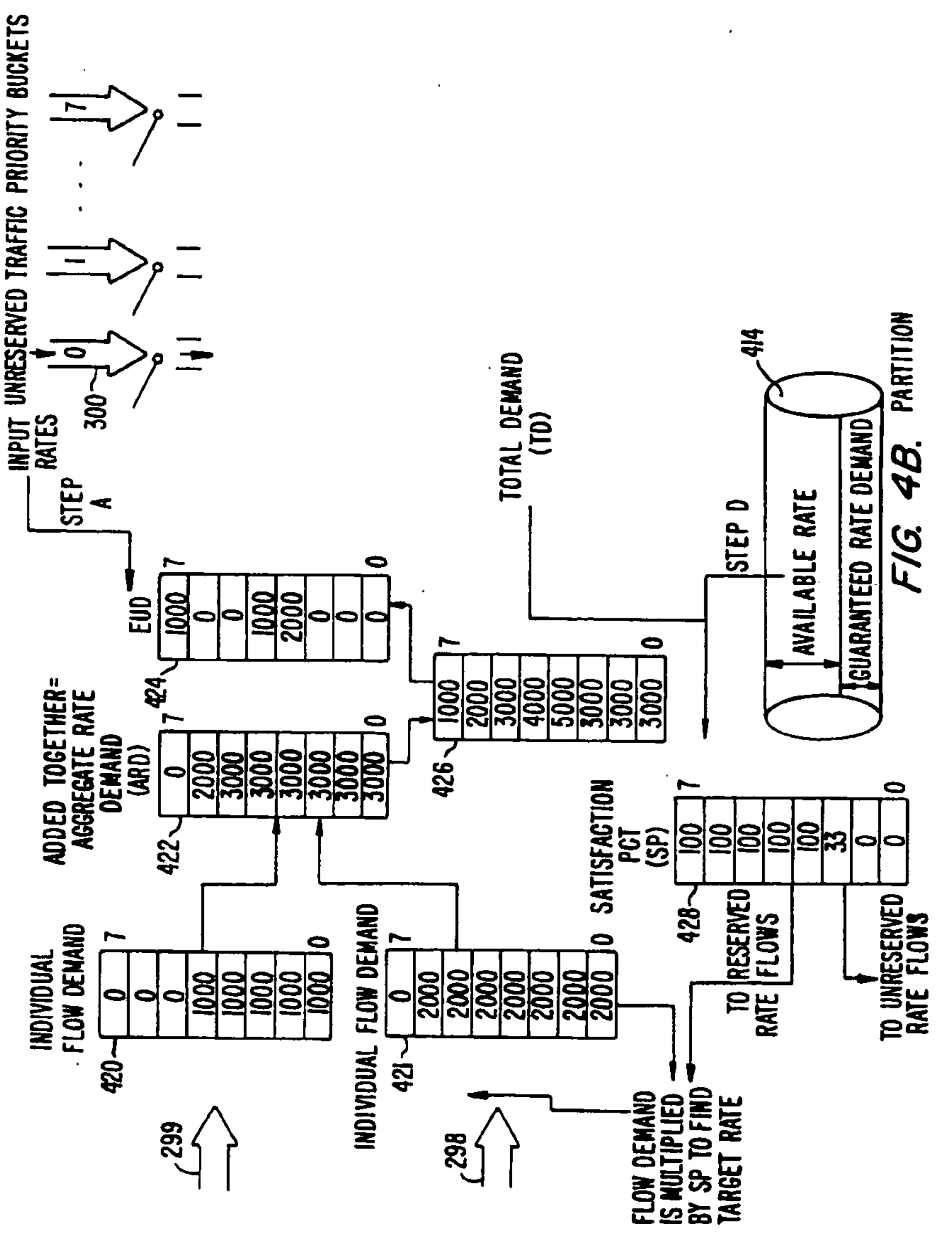

INPUT UNRESERVED TRAFFIC PRIORITY BUCKETS
RATES
STEP A
300
0
1
7
EUD
424
1000 0 0 1000 2000 0 0 0
ADDED TOGETHER= AGGREGATE RATE DEMAND (ARD)
422
0 2000 3000 3000 3000 3000 3000 3000
426
1000 2000 3000 4000 5000 3000 3000 3000
TOTAL DEMAND (TD)
STEP D
AVAILABLE RATE
GUARANTEED RATE DEMAND
414
PARTITION
INDIVIDUAL FLOW DEMAND
420
0 0 0 1000 1000 1000 1000 1000
INDIVIDUAL FLOW DEMAND
421
0 2000 2000 2000 2000 2000 2000 2000
SATISFACTION PCT (SP)
428
100 100 100 100 100 33 0 0
TO RESERVED RATE FLOWS
TO UNRESERVED RATE FLOWS
FLOW DEMAND IS MULTIPLIED BY SP TO FIND TARGET RATE
299
298

FIG. 4B.

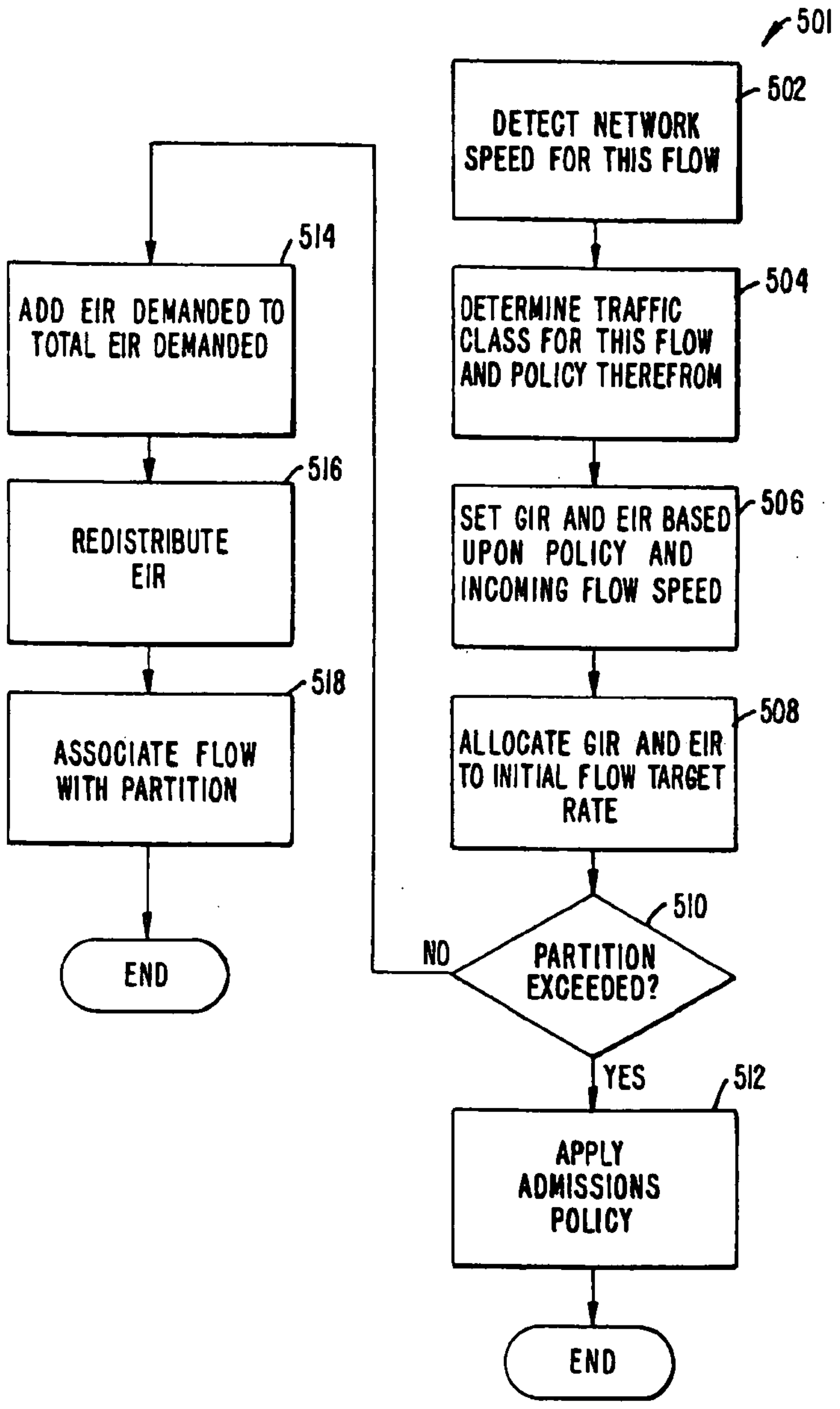

501
502
DETECT NETWORK SPEED FOR THIS FLOW
504
DETERMINE TRAFFIC CLASS FOR THIS FLOW AND POLICY THEREFROM
506
SET GIR AND EIR BASED UPON POLICY AND INCOMING FLOW SPEED
508
ALLOCATE GIR AND EIR TO INITIAL FLOW TARGET RATE
510
PARTITION EXCEEDED?
NO
YES
512
APPLY ADMISSIONS POLICY
END
514
ADD EIR DEMANDED TO TOTAL EIR DEMANDED
516
REDISTRIBUTE EIR
518
ASSOCIATE FLOW WITH PARTITION
END

FIG. 5A.

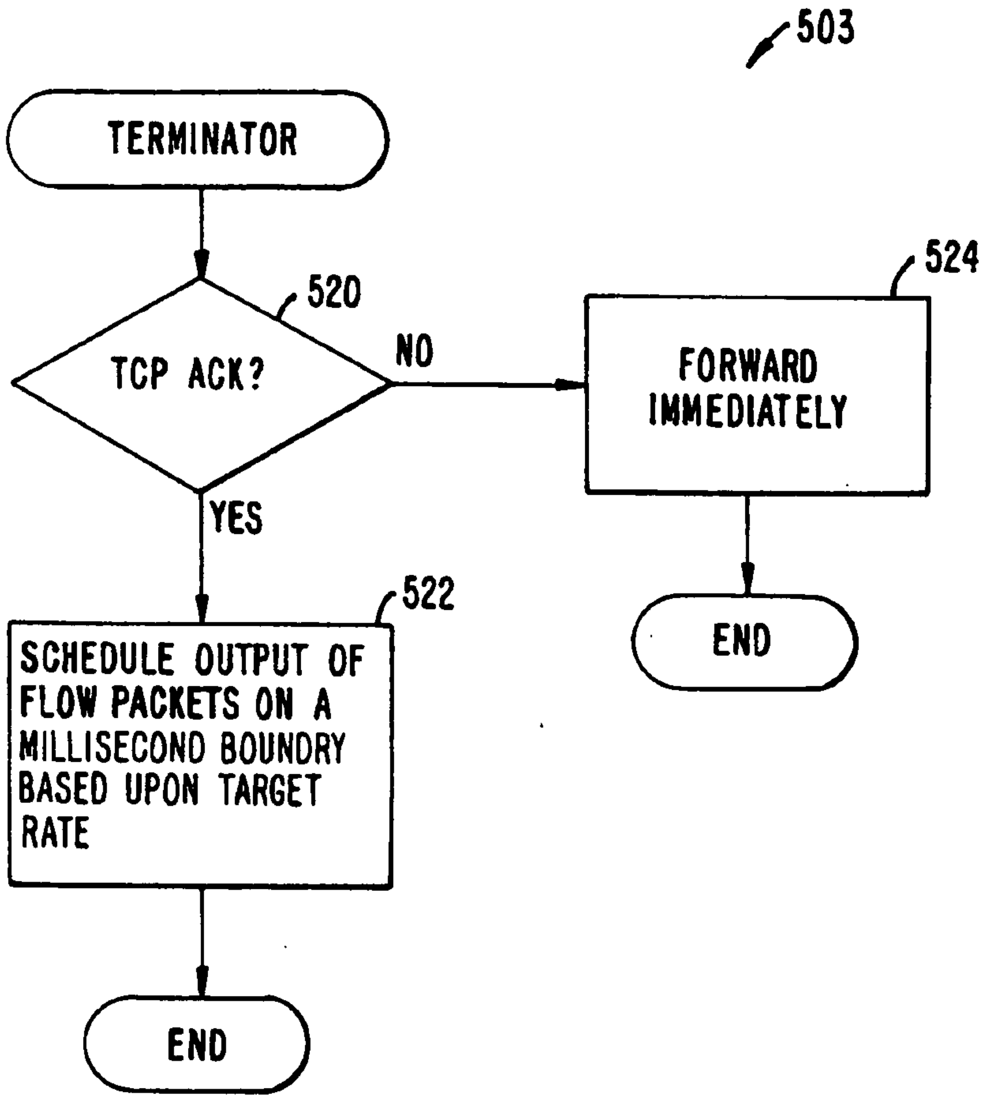

503
TERMINATOR
520
TCP ACK?
NO
YES
524
FORWARD IMMEDIATELY
END
522
SCHEDULE OUTPUT OF FLOW PACKETS ON A MILLISECOND BOUNDRY BASED UPON TARGET RATE
END

FIG. 5B.

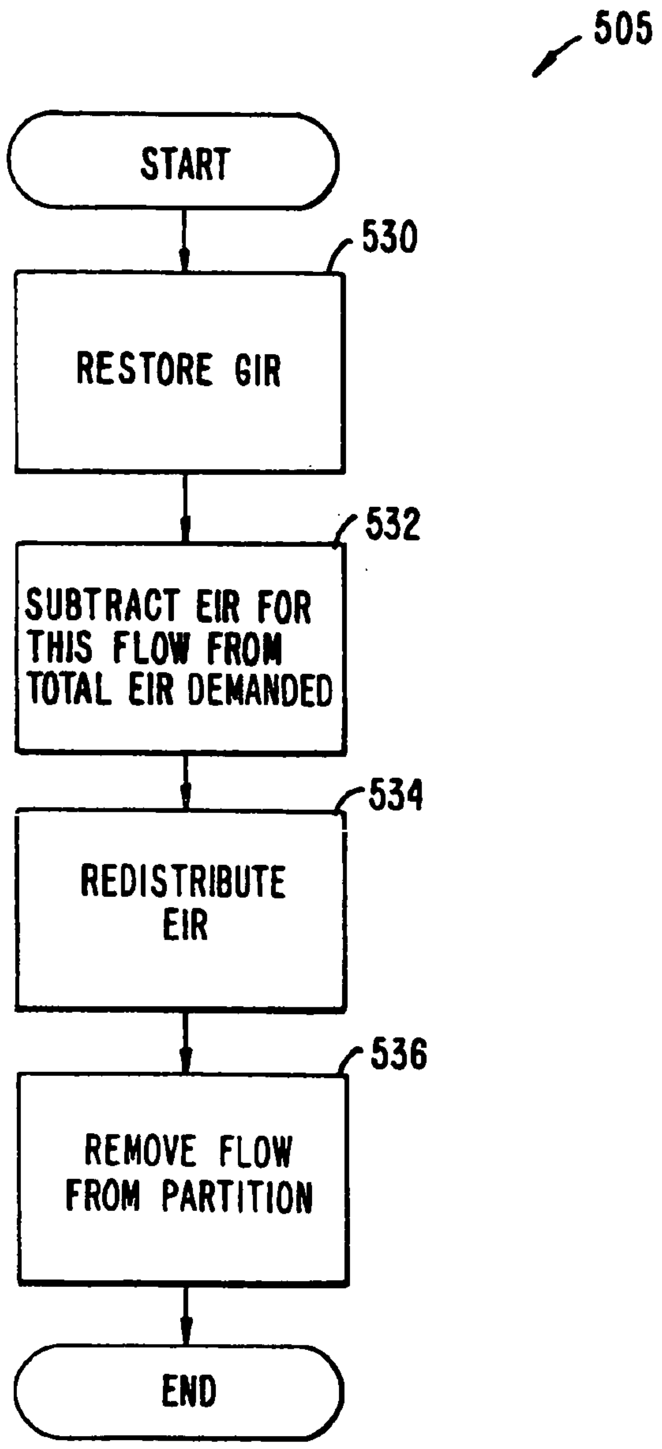

505
START
530
RESTORE GIR
532
SUBTRACT EIR FOR THIS FLOW FROM TOTAL EIR DEMANDED
534
REDISTRIBUTE EIR
536
REMOVE FLOW FROM PARTITION
END

FIG. 5C.

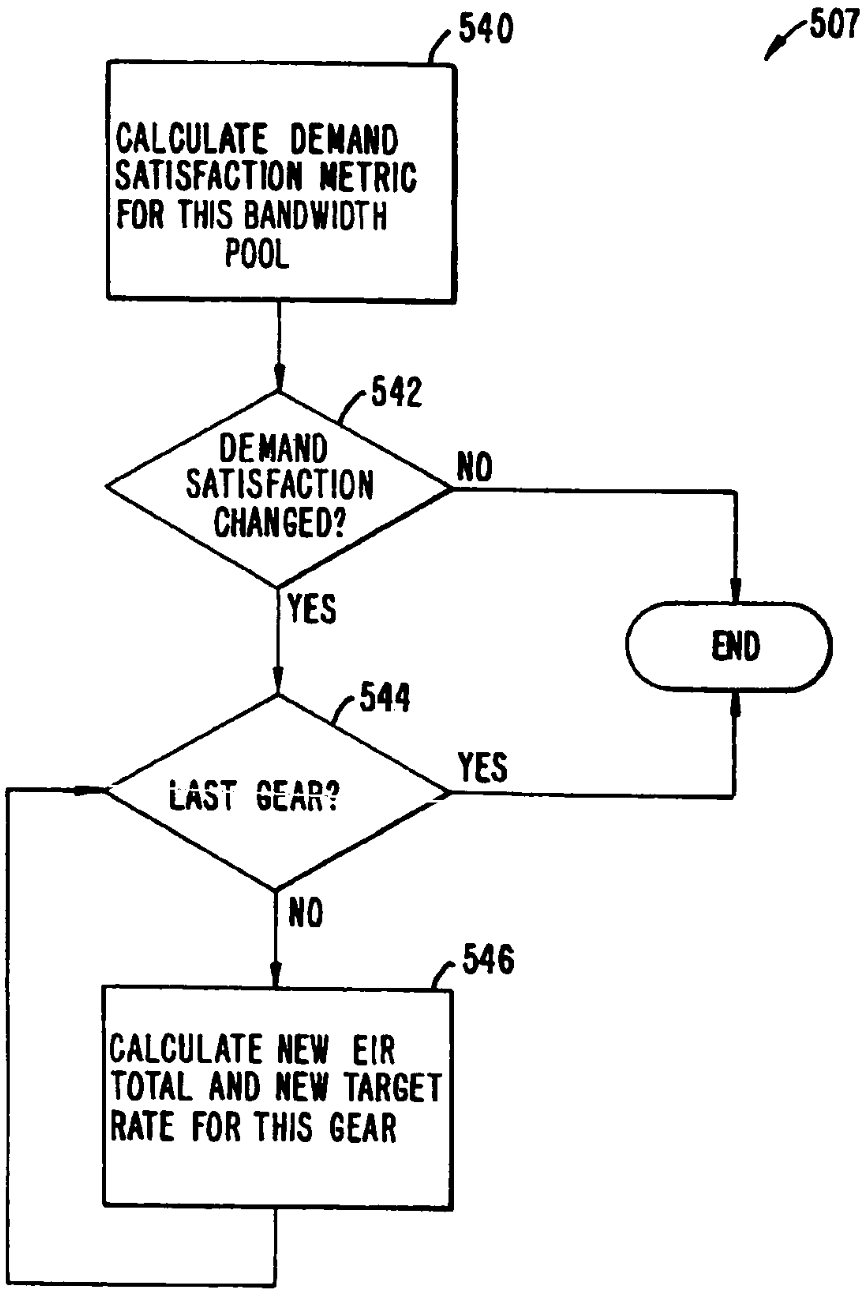

507
540
CALCULATE DEMAND SATISFACTION METRIC FOR THIS BANDWIDTH POOL
542
DEMAND SATISFACTION CHANGED?
NO
YES
END
544
LAST GEAR?
YES
NO
546
CALCULATE NEW EIR TOTAL AND NEW TARGET RATE FOR THIS GEAR

FIG. 5D.

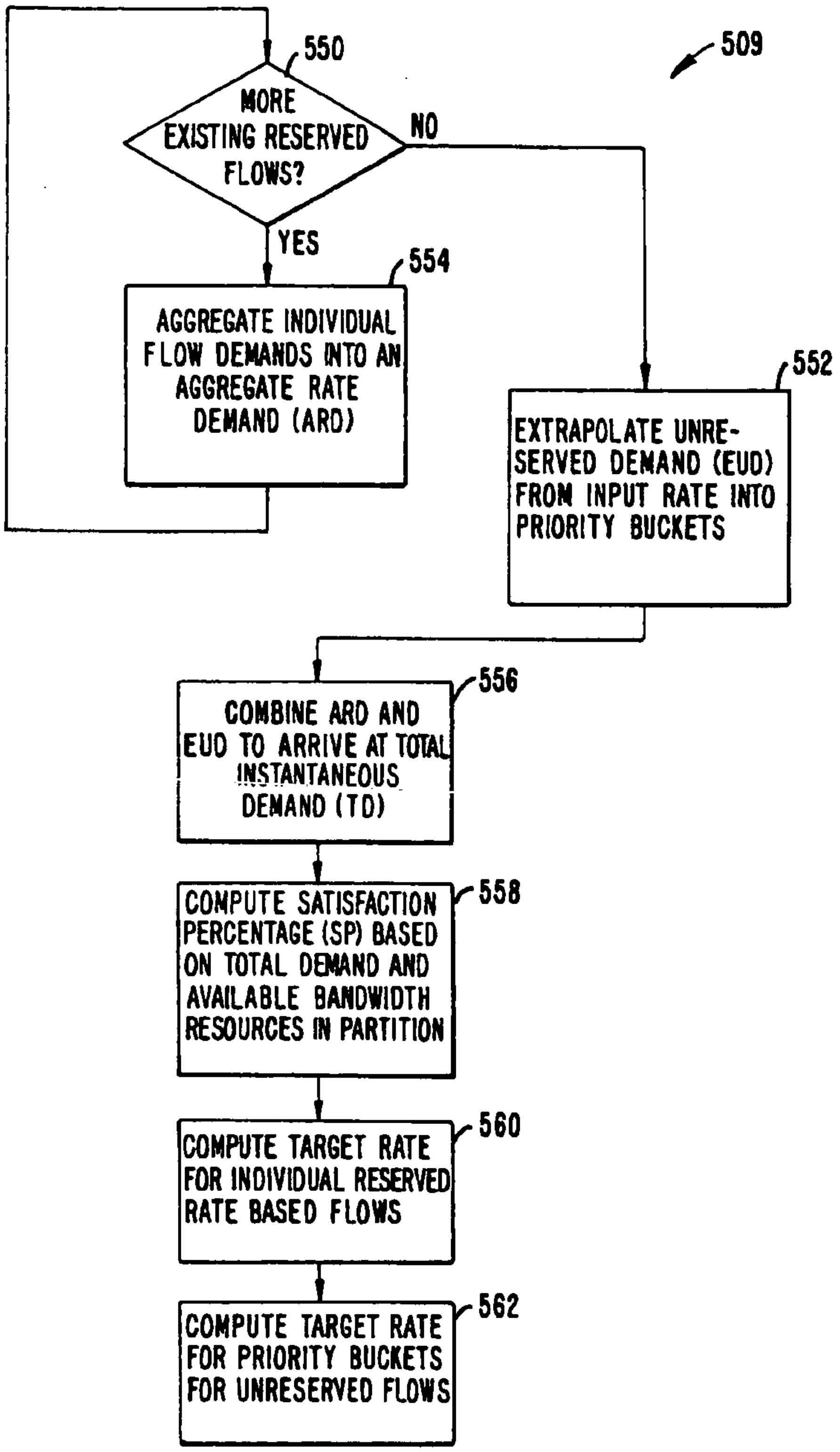

509
550
MORE EXISTING RESERVED FLOWS?
NO
YES
554
AGGREGATE INDIVIDUAL FLOW DEMANDS INTO AN AGGREGATE RATE DEMAND (ARD)
552
EXTRAPOLATE UNRESERVED DEMAND (EUD) FROM INPUT RATE INTO PRIORITY BUCKETS
556
COMBINE ARD AND EUD TO ARRIVE AT TOTAL INSTANTANEOUS DEMAND (TD)
558
COMPUTE SATISFACTION PERCENTAGE (SP) BASED ON TOTAL DEMAND AND AVAILABLE BANDWIDTH RESOURCES IN PARTITION
560
COMPUTE TARGET RATE FOR INDIVIDUAL RESERVED RATE BASED FLOWS
562
COMPUTE TARGET RATE FOR PRIORITY BUCKETS FOR UNRESERVED FLOWS

FIG. 5E.

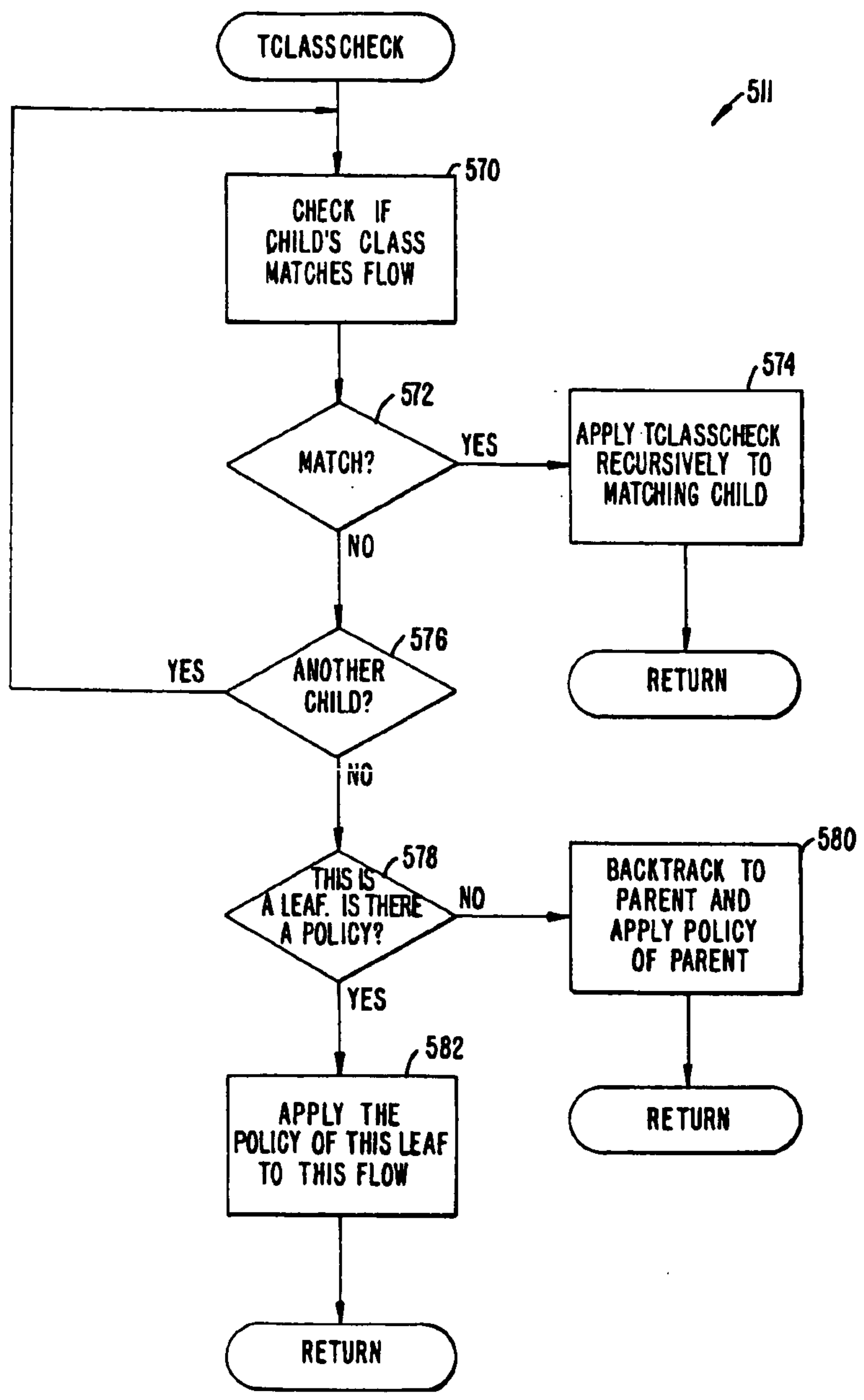

TCLASSCHECK
511
570
CHECK IF CHILD'S CLASS MATCHES FLOW
572
MATCH?
YES
574
APPLY TCLASSCHECK RECURSIVELY TO MATCHING CHILD
RETURN
NO
576
ANOTHER CHILD?
YES
NO
578
THIS IS A LEAF. IS THERE A POLICY?
NO
580
BACKTRACK TO PARENT AND APPLY POLICY OF PARENT
RETURN
YES
582
APPLY THE POLICY OF THIS LEAF TO THIS FLOW
RETURN

FIG. 5F.

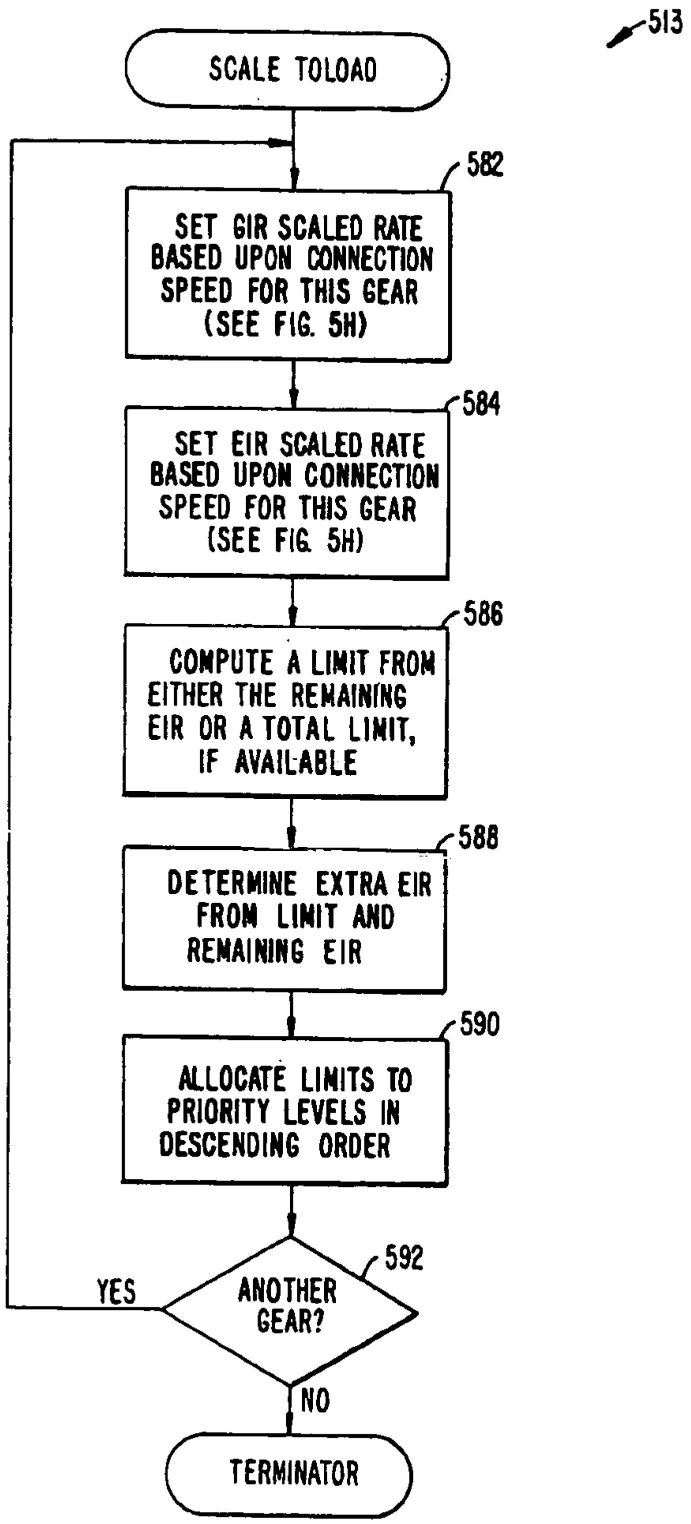

513
SCALE TOLOAD
582
SET GIR SCALED RATE BASED UPON CONNECTION SPEED FOR THIS GEAR (SEE FIG. 5H)
584
SET EIR SCALED RATE BASED UPON CONNECTION SPEED FOR THIS GEAR (SEE FIG. 5H)
586
COMPUTE A LIMIT FROM EITHER THE REMAINING EIR OR A TOTAL LIMIT, IF AVAILABLE
588
DETERMINE EXTRA EIR FROM LIMIT AND REMAINING EIR
590
ALLOCATE LIMITS TO PRIORITY LEVELS IN DESCENDING ORDER
592
ANOTHER GEAR?
YES
NO
TERMINATOR

FIG. 5G.
(STEP 506 OF FIG. 5A)

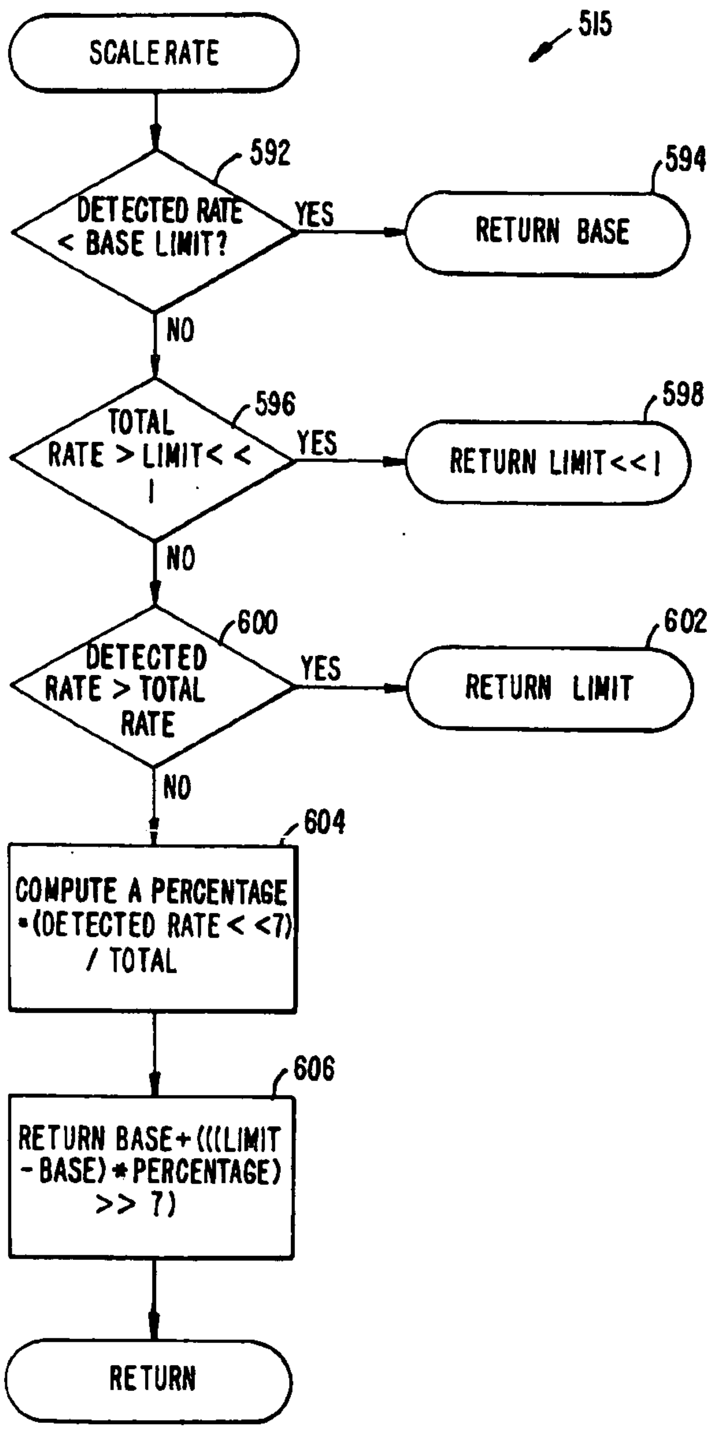

515
SCALE RATE
592
DETECTED RATE < BASE LIMIT?
YES
594
RETURN BASE
NO
596
TOTAL RATE > LIMIT< < 1
YES
598
RETURN LIMIT<<1
NO
600
DETECTED RATE > TOTAL RATE
YES
602
RETURN LIMIT
NO
604
COMPUTE A PERCENTAGE =(DETECTED RATE < <7) / TOTAL
606
RETURN BASE+(((LIMIT - BASE) * PERCENTAGE) >> 7)
RETURN

FIG. 5H.
(STEPS 582 AND 584 OF FIG. 5G)

SYSTEM FOR MANAGING FLOW BANDWIDTH UTILIZATION AT NETWORK, TRANSPORT AND APPLICATION LAYERS IN STORE AND FORWARD NETWORK

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of Ser. No. 08/977,642 filed on Nov. 24, 1997 now U.S. Pat. No. 6,046,980.

The following related commonly-owned copending application is being filed concurrently and is hereby incorporated by reference in its entirety for all purposes: U.S. patent application Ser. No. 08/877,376, now abandoned in the name of Robert L. Packer, entitled "Method for Managing Flow Bandwidth Utilization at Network, Transport and Application Layers,".

This application claims priority from the following U.S. Provisional Application, the disclosure of which, including all appendices and all attached documents, is incorporated by reference in its entirety for all purposes:

U.S. Provisional Patent Application Serial No. 60/032,485, Robert L. Packer, entitled, "Method for Managing Flow Bandwidth Utilization at Network, Transport and Application Layers in Store and Forward Network", filed Dec. 9, 1996.

Further, this application makes reference to the following commonly owned U.S. Patent Application, which is incorporated herein in its entirety for all purposes:

Copending U.S. patent application Ser. No. 08/762,828, now U.S. Pat. No. 5,802,106, in the name of Robert L. Packer, entitled "Method for Rapid Data Rate Detection in a Packet Communication Environment Without Data Rate Supervision," relates to a technique for automatically determining the data rate of a TCP connection.

Further, this application makes reference to the following U.S. Patent Application:

Copending U.S. patent application Ser. No. 08/742,994, now in U.S. Pat. No. 6,038,216, in the name of Robert L. Packer, entitled "Method for Explicit Data Rate Control in a Packet Communication Environment Without a Data Rate Supervision," relates to a technique for automatically scheduling TCP packets for transmission.

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

This invention relates to digital packet telecommunications, and particularly to management of network bandwidth based on information ascertainable from multiple layers of OSI network model. It is particularly useful in conjunction with data flow rate detection and control of a digitally-switched packet telecommunications environment normally not subject to data flow rate control.

The ubiquitous TCP/IP protocol suite, which implements the world-wide data communication network environment called the Internet and is also used in private networks (Intranets), intentionally omits explicit supervisory function

over the rate of data transport over the various media which comprise the network. While there are certain perceived advantages, this characteristic has the consequence of juxtaposing very highspeed packet flows and very low-speed packet flows in potential conflict for network resources, which results in inefficiencies. Certain pathological loading conditions can result in instability, overloading and data transfer stoppage. Therefore, it is desirable to provide some mechanism to optimize efficiency of data transfer while minimizing the risk of data loss. Early indication of the rate of data flow which can or must be supported is very useful. In fact, data flow rate capacity information is a key factor for use in resource allocation decisions.

Internet/Intranet technology is based largely on the TCP/IP protocol suite, where IP, or Internet Protocol, is the network layer protocol and TCP, or Transmission Control Protocol, is the transport layer protocol. At the network level, IP provides a "datagram" delivery service. By contrast, TCP builds a transport level service over the datagram service to provide guaranteed, sequential delivery of a byte stream between two IP hosts.

TCP flow control mechanisms operate exclusively at the end stations to limit the rate at which TCP endpoints emit data. However, TCP lacks explicit data rate control. In fact, there is heretofore no concept of coordination of data rates among multiple flows. The basic TCP flow control mechanism is a sliding window, superimposed on a range of bytes beyond the last explicitly-acknowledged byte. Its sliding operation limits the amount of unacknowledged transmissible data that a TCP endpoint can emit.

Another flow control mechanism is a congestion window, which is a refinement of the sliding window scheme, which employs conservative expansion to fully utilize all of the allowable window. A component of this mechanism is sometimes referred to as "slow start".

The sliding window flow control mechanism works in conjunction with the Retransmit Timeout Mechanism (RTO), which is a timeout to prompt a retransmission of unacknowledged data. The timeout length is based on a running average of the Round Trip Time (RTT) for acknowledgment receipt, i.e. if an acknowledgment is not received within (typically) the smoothed RTT+4*mean deviation, then packet loss is inferred and the data pending acknowledgment is retransmitted.

Data rate flow control mechanisms which are operative end-to-end without explicit data rate control draw a strong inference of congestion from packet loss (inferred, typically, by RTO). TCP end systems, for example, will 'back-off', i.e., inhibit transmission in increasing multiples of the base RTT average as a reaction to consecutive packet loss.

Bandwidth Management in TCP/IP Networks

Conventional bandwidth management in TCP/IP networks is accomplished by a combination of TCP end systems and routers which queue packets and discard packets when certain congestion thresholds are exceeded. The discarded, and therefore unacknowledged, packet serves as a feedback mechanism to the TCP transmitter. (TCP end systems are clients or servers running the TCP transport protocol, typically as part of their operating system.)

The term "bandwidth mananagent" is often used to refer to link level bandwidth management, e.g. multiple line support for Point to Point Protocol (PPP). Link Level bandwidth management is essentially the process of keeping track of all traffic and deciding whether an additional dial line or ISDN channel should be opened or an extraneous one closed. The field of this invention is concerned with network level bandwidth management, i.e. policies to assign available bandwidth from a single logical link to network flows.

Routers support various queuing options. These options are generally Intended to promote fairness and to provide a rough ability to partition and prioritize separate classes of traffic. Configuring these queuing options with any precision or without side effects is in fact very difficult, and in some cases, not possible. Seemingly simple things, such as the length of the queue, have a profound effect on traffic characteristics, Discarding packets as a feedback mechanism to TCP end systems may cause large, uneven delays perceptible to interactive users.

In a copending U.S. patent application Ser. No. 08/742,994, in the name of Robert L. Packer, entitled "Method for Explicit Data Rate Control in a Packet Communication Envirомnent Without Data Rate Supervision," a technique for automatically scheduling TCP packets for transmission is disclosed. Furthermore, in a copending U.S. patent application Ser. No. 08/762,828, in the name of Robert L. Packer, entitled "Method for Rapid Data Rate Detection in a Packet Communication Environment Without Data Rate Supervision," a technique for automatically determining the data rate of a TCP connection is disclosed. While these patent applications teach methods for solving problems associated with scheduling transmissions and for automatically determining a data flow rate on a TCP connection, respectively, there is no teaching in the prior art of methods for explicitly TCP packet traffic based upon information about the flow's characteristics at multiple OSI protocol layers.

Bandwidth management is heretofore not known to employ information contained in the packets corresponding to higher OSI protocol layers, even though such information may be extremely useful in making bandwidth allocation and management decisions.

SUMMARY OF THE INVENTION

According to the invention, in a packet communication environment, a method is provided for classifying packet network flows for use in determining a policy, or rule of assignment of a service level, and enforcing that policy by direct rate control. The method comprises applying individual instances of traffic objects, i.e., packet network flows to a classification model based on selectable information obtained from a plurality of layers of a multi-layered communication protocol, then mapping the flow to the defined traffic classes, which are arbitrarily assignable by an offline manager which creates the classification. It is useful to note that the classification need not be a complete enumeration of the possible traffic.

In one aspect of the invention, bandwidth may be divided into arbitrary units, partitions, facilitating isolation and allocation. A partition is allocated for a class set of traffic classes, carving the bandwidth of the associated link into multiple, independent pieces.

In another aspect of the invention, available bandwidth may be allocated among flows according to a policy, which may include any combination of guaranteed information rate, excess information rate, the later allocated according to a priority.

In another aspect of the invention, bandwidth resource needs of multiple heterogeneous requesting flows are reconciled with available bandwidth resources in accordance with policy of each flow based upon the flow's class. Flows requiring reserved service with guaranteed information rates, excess information rates or unreserved service are reconciled with the available bandwidth resources continuously and automatically.

In another aspect of the invention, providing an admissions policy which is invoked whenever a request for a bandwidth cannot be met consistently with other users of bandwidth.

An advantage of network management techniques according to the present invention is that network managers need only define traffic classes which are of interest.

A further advantage of the present invention is that traffic classes may include information such as a URI for web traffic.

A yet further advantage of the present invention is that service levels may be defined in terms of explicit rates and may be scaled to a remote client or server's network access rate. Different service levels may be specified for high speed and low speed users.

A yet further advantage of the present invention is that service levels may be defined in terms of a guaranteed minimum service level.

The invention will be better understood upon reference to the following detailed description in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A depicts a representative client server relationship in accordance with a particular embodiment of the invention;

FIG. 1B depicts a functional perspective of the representative client server relationship in accordance with a particular embodiment of the invention;

FIG. 1C depicts a representative internetworking environment in accordance with a particular embodiment of the invention;

FIG. 1D depicts a relationship diagram of the layers of the TCP/IP protocol suite;

FIG. 1E depicts a two dimensional representation of timing relationships in the exchange of packets between hosts using the TCP protocol;

FIGS. 2A–2B depict representative divisions of bandwidth according to a particular embodiment of the invention;

FIGS. 2C–2E are flow charts depicting process steps according to a particular embodiment of the invention;

FIG. 3 is a block diagram of a particular embodiment according to the invention;

FIG. 4A is a block diagram of a data structure according to a particular embodiment of the invention;

FIG. 4B is a block diagram of data structure according to a particular embodiment of the invention; and

FIGS. 5A–5H are flow charts depicting process steps according to a particular embodiment of the invention.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

A preferable embodiment of a flow bandwidth management system according to the invention has been reduced to practice and will be made available under the trade name "PacketShaper™."

1.0 Introduction

The present invention provides techniques to manage network bandwidth such as on a network access link between a local area network and a wide area network.

Systems according to the present invention enable network managers to: define traffic classes; create policies which define service levels for traffic classes; and isolate bandwidth resources associated with certain traffic classes. Inbound as well as outbound traffic may be managed. Table 1 provides a definitional list of terminology used herein.

TABLE 1

LIST OF DEFINITIONAL TERMS	
ADMISSIONS CONTROL	A policy invoked whenever a system according to the invention detects that a guaranteed information rate cannot be maintained. An admissions control policy is analogous to a busy signal in the telephone world.
CLASS SEARCH ORDER	A search method based upon traversal of a N-ary tree data structure containing classes.
COMMITTED INFORMATION RATE (CIR)	A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for a committed bandwidth. Also called a guaranteed information rate (GIR).
EXCEPTION	A class of traffic provided by the user which supersedes an automatically determined classification order.
EXCESS INFORMATION RATE (EIR)	A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for uncommitted bandwidth resources.
FLOW	A flow is a single instance of a traffic class. For example, all packets in a TCP connection belong to the same flow. As do all packets in a UDP session.
GUARANTEED INFORMATION RATE (GIR)	A rate of data flow allocated to reserved service traffic for rate based bandwidth allocation for a committed bandwidth. Also called a committed information rate (CIR).
HARD ISOLATION	Hard isolation results from the creation of an entirely separate logical channel for a designated set of classes.
INSIDE	On the system side of an access link. Outside clients and servers are on the other side of the access link.
ISOLATION	Isolation is the degree that bandwidth resources are allocable to traffic classes.
OUTSIDE	On the opposite side of an access link as viewed from the perspective of the system on which the software resides.
PARTITION	Partition is an arbitrary unit of network resources.
POLICY	A rule for the assignment of a service level to a flow.
POLICY INHERITANCE	A method for assigning policies to flows for which no policy exists in a hierarchical arrangement of policies. For example, if a flow is determined to be comprised of FTP packets for Host A, and no corresponding policy exists, a policy associated with a parent node, such as an FTP policy, may be located and used. See also POLICY SEARCH ORDER.
POLICY BASED SCALING	An adjustment of a requested data rate for a particular flow based upon the policy associated with the flow and information about the flow's potential rate.
RESERVED SERVICE	Reserved service is a service level intended for traffic which "bursts" or sends chunks of data. Reserved service is defined in terms of a scaled rate.
SCALED RATE	Assignment of a data rate based upon detected speed.
SERVICE LEVEL	A service paradigm having a combination of characteristics defined by a network manager to handle a particular class of traffic. Service levels may be designated as either reserved or unreserved.
SOFT ISOLATION	Restricting GIR allocated for traffic classes in a partition.

TABLE 1-continued

LIST OF DEFINITIONAL TERMS	
TARGET RATE	A target rate is a combination of a guaranteed rate and an excess rate. Target rate is a policy-based paradigm. Excess rate is allocated by systems according to the invention from bandwidth that is not consumed by reserved service. Policies will demand excess rate at a given priority and systems according to the invention satisfy this demand by a priority level.
TRAFFIC CLASS	All traffic between a client and a server endpoints. A single instance of a traffic class is called a flow. Traffic classes have properties or class attributes such as, directionality, which is the property of traffic to be flowing inbound or outbound.
UNRESERVED SERVICE	Unreserved service is a service level defined in terms of priority in which no reservation of bandwidth is made.

1.1 Hardware Overview

The method for flow bandwidth management in a packet oriented telecommunications network environment of the present invention is implemented in the C programming language and is operational on a computer system such as shown in FIG. 1A. This invention may be implemented in a client-server environment, but a client-server environment is not essential. This figure shows a conventional client-server computer system which includes a server 20 and numerous clients, one of which is shown as client 25. The use of the term "server" is used in the context of the invention, wherein the server receives queries from (typically remote) clients, does substantially all the processing necessary to formulate responses to the queries, and provides these responses to the clients. However, server 20 may itself act in the capacity of a client when it accesses remote databases located at another node acting as a database server.

The hardware configurations are in general standard and will be described only briefly. In accordance with known practice, server 20 includes one or more processors 30 which communicate with a number of peripheral devices via a bus subsystem 32. These peripheral devices typically include a storage subsystem 35, comprised of a memory subsystem 35*a* and a file storage subsystem 35*b* holding computer programs (e.g., code or instructions) and data, a set of user interface input and output devices 37, and an interface to outside networks, which may employ Ethernet, Token Ring, ATM, IEEE 802.3, ITU X.25, Serial Link Internet Protocol (SLIP) or the public switched telephone network. This interface is shown schematically as a "Network Interface" block 40. It is coupled to corresponding interface devices in client computers via a network connection 45.

Client 25 has the same general configuration, although typically with less storage and processing capability. Thus, while the client computer could be a terminal or a low-end personal computer, the server computer is generally a high-end workstation or mainframe, such as a SUN SPARC server. Corresponding elements and subsystems in the client computer are shown with corresponding, but primed, reference numerals.

Bus subsystem 32 is shown schematically as a single bus, but a typical system has a number of buses such as a local bus and one or more expansion buses (e.g., ADB, SCSI, ISA, EISA, MCA, NuBus, or PCI), as well as serial and parallel ports. Network connections are usually established through a device such as a network adapter on one of these expansion buses or a modem on a serial port. The client computer may be a desktop system or a portable system.

The user interacts with the system using interface devices 37' (or devices 37 in a standalone system). For example, client queries are entered via a keyboard, communicated to client processor 30', and thence to modem or network interface 40' over bus subsystem 32'. The query is then communicated to server 20 via network connection 45. Similarly, results of the query are communicated from the server to the client via network connection 45 for output on one of devices 37' (say a display or a printer), or may be stored on storage subsystem 35'.

FIG. 1B is a functional diagram of a computer system such as that of FIG. 1A. FIG. 1B depicts a server 20, and a representative client 25 of a plurality of clients which may interact with the server 20 via the Internet 45 or any other communications method. Blocks to the right of the server are indicative of the processing steps and functions which occur in the server's program and data storage indicated by blocks 35*a* and 35*b* in FIG. 1A. A TCP/IP "stack" 44 works in conjunction with Operating System 42 to communicate with processes over a network or serial connection attaching Server 20 to Internet 45 . Web server software 46 executes concurrently and cooperatively with other processes in server 20 to make data objects 50 and 51 available to requesting clients. A Common Gateway Interface (CGI) script 55 enables information from user clients to be acted upon by web server 46 , or other processes within server 20. Responses to client queries may be returned to the clients in the form of a Hypertext Markup Language (HTML) document outputs which are then communicated via Internet 45 back to the user.

Client 25 in FIG. 1B possesses software implementing functional processes operatively disposed in its program and data storage as indicated by block 35*a*' in FIG. 1A. TCP/IP stack 44', works in conjunction with Operating System 42' to communicate with processes over a network or serial connection attaching Client 25 to Internet 45. Software implementing the function of a web browser 46' executes concurrently and cooperatively with other processes in client 25 to make requests of server 20 for data objects 50 and 51. The user of the client may interact via the web browser 46' to make such queries of the server 20 via Internet 45 and to view responses from the server 20 via Internet 45 on the web browser 46'.

1.2 Network Overview

FIG. 1C is illustrative of the internetworking of a plurality of clients such as client 25 of FIGS. 1A and 1B and a plurality of servers such as server 20 of FIGS. 1A and 1B as described herein above. In FIG. 1C, network 70 is an example of a Token Ring or frame oriented network. Network 70 links host 71, such as an IBM RS6000 RISC workstation, which may be running the AIX operating system, to host 72, which is a personal computer, which may be running Windows 95, IBM OS/2 or a DOS operating system, and host 73, which may be an IBM AS/400 computer, which may be running the OS/400 operating system. Network 70 is internetworked to network 60 via a system gateway which is depicted here as router 75, but which may also be a gateway having a firewall or a network bridge. Network 60 is an example of an Ethernet network that interconnects host 61, which is a SPARC workstation, which may be running SUNOS operating system with host 62, which may be a Digital Equipment VAX6000 computer which may be running the VMS operating system.

Router 75 is a network access point (NAP) of network 70 and network 60. Router 75 employs a Token Ring adapter and Ethernet adapter. This enables router 75 to interface with the two heterogeneous networks. Router 75 is also aware of the Inter-network Protocols, such as ICMP ARP and RIP, which are described herein below.

FIG. 1D is illustrative of the constituents of the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol suite. The base layer of the TCP/IP protocol suite is the physical layer 80, which defines the mechanical, electrical, functional and procedural standards for the physical transmission of data over communications media, such as, for example, the network connection 45 of FIG. 1A. The physical layer may comprise electrical, mechanical or functional standards such as whether a network is packet switching or frame-switching; or whether a network is based on a Carrier Sense Multiple Access/Collision Detection (CSMA/CD) or a frame relay paradigm.

Overlying the physical layer is the data link layer 82. The data link layer provides the function and protocols to transfer data between network resources and to detect errors that may occur at the physical layer. Operating modes at the datalink layer comprise such standardized network topologies as IEEE 802.3 Ethernet, IEEE 802.5 Token Ring, ITU X.25, or serial (SLIP) protocols.

Network layer protocols 84 overlay the datalink layer and provide the means for establishing connections between networks. The standards of network layer protocols provide operational control procedures for internetworking communications and routing information through multiple heterogenous networks. Examples of network layer protocols are the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP). The Address Resolution Protocol (ARP) is used to correlate an Internet address and a Media Access Address (MAC) for a particular host. The Routing Information Protocol (RIP) is a dynamic routing protocol for passing routing information between hosts on networks. The Internet Control Message Protocol (ICMP) is an internal protocol for passing control messages between hosts on various networks. ICMP messages provide feedback about events in the network environment or can help determine if a path exists to a particular host in the network environment. The latter is called a "Ping". The Internet Protocol (IP) provides the basic mechanism for routing packets of information in the Internet. IP is a non-reliable communication protocol. It provides a "best efforts" delivery service and does not commit network resources to a particular transaction, nor does it perform retransmissions or give acknowledgments.

The transport layer protocols 86 provide end-to-end transport services across multiple heterogenous networks. The User Datagram Protocol (UDP) provides a connectionless, datagram oriented service which provides a non-reliable delivery mechanism for streams of information. The Transmission Control Protocol (TCP) provides a reliable session-based service for delivery of sequenced packets of information across the Internet. TCP provides a connection oriented reliable mechanism for information delivery.

The session, or application layer 88 provides a list of network applications and utilities, a few of which are illustrated here. For example, File Transfer Protocol (FTP) is a standard TCP/IP protocol for transferring files from one machine to another. FTP clients establish sessions through TCP connections with FTP servers in order to obtain files. Telnet is a standard TCP/IP protocol for remote terminal connection. A Telnet client acts as a terminal emulator and establishes a connection using TCP as the transport mechanism with a Telnet server. The Simple Network Management Protocol (SNMP) is a standard for managing TCP/IP networks. SNMP tasks, called "agents", monitor network status parameters and transmit these status parameters to SNMP tasks called "managers." Managers track the status of associated networks. A Remote Procedure Call (RPC) is a programming interface which enables programs to invoke remote functions on server machines. The Hypertext Transfer Protocol (HTTP) facilitates the transfer of data objects across networks via a system of uniform resource indicators (URI).

The Hypertext Transfer Protocol is a simple protocol built on top of Transmission Control Protocol (TCP). It is the mechanism which underlies the function of the World Wide Web. The HTTP provides a method for users to obtain data objects from various hosts acting as servers on the Internet. User requests for data objects are made by means of an HTTP request, such as a GET request. A GET request as depicted below is comprised of 1) an HTTP protocol version, such as "http:/1.0"; followed by 2) the full path of the data object; followed by 3) the name of the data object. In the GET request shown below, a request is being made for the data object with a path name of "/pub/" and a name of "MyData.html":

HTTP-Version GET /pub/MyData.html (1)

Processing of a GET request entails the establishing of an TCP/IP connection with the server named in the GET request and receipt from the server of the data object specified. After receiving and interpreting a request message, a server responds in the form of an HTTP RESPONSE message.

Response messages begin with a status line comprising a protocol version followed by a numeric Status Code and an associated textual Reason Phrase. These elements are separated by space characters. The format of a status line is depicted in line (2):

Status-Line=HTTP-Version Status-Code Reason-Phrase (2)

The status line always begins with a protocol version and status code, e.g., "HTTP/1.0 200". The status code element is a three digit integer result code of the attempt to understand and satisfy a prior request message. The reason phrase is intended to give a short textual description of the status code.

The first digit of the status code defines the class of response. There are five categories for the first digit. 1XX is an information response. It is not currently used. 2XX is a successful response, indicating that the action was successfully received, understood and accepted. 3XX is a redirection response, indicating that further action must be taken in order to complete the request. 4XX is a client error response. This indicates a bad syntax in the request. Finally, 5XX is a server error. This indicates that the server failed to fulfill an apparently valid request.

Particular formats of HTTP messages are described in, Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982, which is incorporated by reference herein for all purposes.

1.3 TCP Data Link Rate and Latency Period Determination

Techniques for determining data rates are more fully set forth in co-owned U.S. patent application Ser. No. 08/762,828, entitled "Method for Rapid Data Rate Detection in a Packet Communication Environment Without Data Rate Supervision", which is incorporated herein by reference for all purposes.

FIG. 1E depicts a two-dimensional timing diagram illustrative of one particular method for determining a data link rate and latency period from an exchange of packets between TCP endpoints. According to this method, the initial data packets are examined as they establish a connection. Parameters are developed from which round trip time (RTT) and maximum data rate can be determined. The serialization speed (SS) or data flow rate capacity of a link is given by the relation:

$$SS=m/T_1 \qquad (3)$$

where:

m=number of bytes in the first data packet (Data1)

T_1=The arrival time of the first data packet less the time of arrival of the ACK packet ($T_{data1}-T_{ACK}$)

FIG. 1E shows a two dimensional timing diagram of the connection protocol of a remote HTTP request between a local TCP endpoint (server) and a remote TCP endpoint (client). The remote endpoint issues a request for connection in the form of a SYN packet in step A. The SYN packet takes a finite but unknown transit time to arrive at the local TCP endpoint. The local TCP endpoint responds by sending its own SYN packet in step B. This SYN packet is of a known byte length and is issued at a known time, which becomes the reference time, "tbase." After a brief latency, the remote TCP endpoint issues a standard ACK packet, whose length is likewise known, in step C, and then also issues the first data packet data1 in step D.

Time T_1, is computed immediately at the time of arrival of data1 by examining the difference in arrival time of the received ACK packet and the data1 packet. The value of m is extracted by examining the entire first packet to determine its length or by examining the packet length field, if any, in the header information of the packet. A first estimate of serialization speed SS is given by Equation (3). Serialization speed can be estimated immediately upon interchange of the first packets and used to make important strategic decisions about the nature and speed of the connection about to be established.

An alternative method for determining a data link rate and latency period from an exchange of packets between TCP endpoints develops parameters from which round trip time (RTT) and maximum data rate can be determined. Values are obtained for serialization of n, the size (i.e., data length) of the SYN packet in response, plus the size (i.e., data length) of the ACK packet. Serialization time is determined according to equation (4) by dividing the value n by the transit time T_0, or

$$SS(\text{max possible})=n/T_0 \qquad (4)$$

where

n=number of bytes in the SYN packet plus the number of bytes in the ACK packet and

T_0=the arrival time of the ACK packet less the tbase value. The round trip time minus the serialization time is the round trip latency T_d.

2.0 Traffic Class

A traffic class is broadly defined as traffic between one or more clients and one or more servers. A single instance of a traffic class is called a flow. Traffic classes have the property, or class attribute, of being directional, i.e. all traffic flowing inbound will belong to different traffic classes and be managed separately from traffic flowing outbound. The directional property enables asymmetric classification and control of traffic, i.e., inbound and outbound flows belong to different classes which may be managed independent of one another.

Traffic classes may be defined at any level of the TCP/IP protocol. For example, at the IP level, traffic may be defined as only those flows between a set of inside and outside IP addresses or domain names. An example of such a low level traffic class definition would be all traffic between my network and other corporate offices throughout the Internet. At the application level, traffic classes may be defined for specific URIs within a web server. Traffic classes may be defined having "Web aware" class attributes. For example, a traffic class could be created such as all URIs matching "*.html" for all servers, or all URIs matching "*.gif" for server X, or for access to server Y with URI "/sales/*" from client Z, wherein '*' is a wildcard character, i.e., a character which matches all other character combinations. Traffic class attributes left unspecified will simply match any value for that attribute. For example, a traffic class that accesses data objects within a certain directory path of a web server is specified by a URI of the directory path to be managed, e.g. "/sales/*".

2.1 Classifying Traffic

The present invention provides a method for classifying traffic according to a definable set of classification attributes selectable by the manager, including selecting a subset of traffic of interest to be classified. The invention provides the ability to classify and search traffic based upon multiple orthogonal classification attributes. Traffic class membership may be hierarchical. Thus, a flow may be classified by a series of steps through a traffic class tree, with the last step (i.e., at the leaves on the classification tree) mapping the flow to a policy. The policy is a rule of assignment for flows. For example, the first step in classification may be to classify a flow as web traffic, the next may further classify this flow as belonging to server X, and the final classification may be a policy for URI "*.avi".

A classification tree is a data structure representing the hierarchical aspect of traffic class relationships. Each node of the classification tree represents a class, and has a traffic specification, i.e., a set of attributes or characteristics describing the traffic, and a mask associated with it. Leaf nodes of the classification tree contain policies. According to a particular embodiment, the classification process checks at each level if the flow being classified matches the attributes of a given traffic class. If it does, processing continues down to the links associated with that node in the tree. If it does not, the class at the level that matches determines the policy for the flow being classified. If no policy specific match is found, the flow is assigned the default policy.

In a preferable embodiment, the classification tree is an N-ary tree with its nodes ordered by specificity. For example, in classifying a particular flow in a classification tree ordered first by organizational departments, the attributes of the flow are compared with the traffic specification in each successive department node and if no match is found, then processing proceeds to the next subsequent department node. If no match is found, then the final compare is a default "match all" category. If, however, a match is found, then classification moves to the children of this department node. The child nodes may be ordered by an orthogonal paradigm such as, for example, "service type." Matching proceeds according to the order of specificity in the child nodes. Processing proceeds in this manner, traversing downward and from left to right in the classification tree, searching the plurality of orthogonal paradigms. Key to implementing this a hierarchy is that the nodes are arranged in decreasing order of specificity. This permits search to find the most specific class for the traffic before more general.

Table 2 depicts components from which Traffic classes may be built. Note that the orientation of the server (inside or outside) is specified. And as noted above, any traffic class component may be unspecified, i.e. set to match any value.

TABLE 2

Traffic Class Components

Client Side	Server Side
IP Address/Domain Name	IP Address/Domain Name TCP or UDP Service, e.g. WWW, FTP, RealAudio, etc. URI for Web Service, e.g. "*.html", "*.gif", "/sales/*", etc.

FIGS. 2A and 2B depict representative allocations of bandwidth made by a hypothetical network manager as an example. In FIG. 2A, the network manager has decided to divide her network resources first by allocating bandwidth between Departments A and B. FIG. 2A shows the resulting classification tree, in which Department A bandwidth resources 202 and Department B bandwidth resources 204 each have their own nodes representing a specific traffic class for that department. Each traffic class may have a policy attribute associated with it. For example, in FIG. 2A, the Department A resources node 202 has the policy attribute Inside-Host Subnet A associated with it. Next, the network manager has chosen to divide the bandwidth resources of Department A among two applications. She allocates an FTP traffic class 206 and a World Wide Web server traffic class 208. Each of these nodes may have a separate policy attribute associated with them. For example, in FIG. 2A, the FTP node 206 for has an attribute Outside port 20 associated with it. Similarly, the network manager has chosen to divide network bandwidth resources of Department B into an FTP server traffic class 210 and a World Wide Web server traffic class 212. Each may have their own respective policies.

FIG. 2B shows a second example, wherein the network manager has chosen to first divide network bandwidth resource between web traffic and TCP traffic. She creates three traffic nodes, a web traffic node 220, a TCP traffic node 224 and a default node 225. Next, she divides the web traffic among two organizational departments by creating a Department A node 226, and a Department B node 228. Each may have its own associated policy. Similarly, she divides TCP network bandwidth into separate traffic classes by creating a Department A node 230 and a Department B node 232. Each represents a separate traffic class which may have its own policy.

All traffic which does not match any user specified traffic class falls into an automatically created default traffic class which has a default policy. In a preferable embodiment, the default policy treats default class flows as unreserved traffic at the default (medium) priority. In FIG. 2A, the default category is depicted by a default node 205, and in FIG. 2B, the default category is depicted by a default node 225.

2.2 Creating Policies

2.2.1 Reserved vs. Unreserved

Network managers create policies in order to assign service levels to traffic classes which are of particular interest. Service levels may be either reserved or unreserved.

Reserved service is useful in instances where traffic 'bursts', i.e., sends chunks of data, or in interactive applications, which require a minimum bandwidth in order to function properly.

Reserved service is defined in terms of scaled rate, unreserved service is defined in terms of priority. Allocation of bandwidth resources by scaled rate is accomplished in a process known as "speed scaling" wherein the speed of the underlying network link upon which the flow is carried is detected and available bandwidth is allocated based upon this detected speed. Allocation of bandwidth resources for unreserved service is priority based. The total bandwidth resources for unreserved service demanded by all flows is satisfied according to a prioritized scheme, in which highest priorities are satisfied first, followed by successively lower priority applications. Unreserved service allocations are based upon the available bandwidth after reserved service demands have been satisfied.

Traffic may be a combination of reserved and unreserved services. For example, web traffic may have a service level defined at a web server for "*.html" to designate these types of files as traffic allocated unreserved service, as these are typically small files. In the same web server, "*.gif" files may be designated as a separate traffic class receiving reserved service. TCP connections and UDP sessions may be defined for reserved service.

2.2.2 Reserved Service: GIR vs. EIR

Reserved service may have both a guaranteed information rate (GIR) and an excess information rate (EIR) components. Guaranteed information rate represents a commitment of bandwidth resources to maintain a specified rate. Excess information rate is an allocation of bandwidth resources on an "as available" basis. Guaranteed information rate prevents evident lack of progress situations, such as stuck progress indicators, from occurring. Excess rate is allocated from available bandwidth, i.e. that which has not consumed by providing guaranteed rate service. As flows demand excess information rate at different priority levels, their demands are satisfied in order of priority level. For example, in an application having a ten voice over IP sessions, each session requiring a minimum of eight kilobits of bandwidth rate which must be guaranteed and up to 8 kilobits of additional bandwidth rate which may be used from time-to-time, would be allocated reserved service comprised of eight kilobits of guaranteed information rate for each of the ten sessions. Then, from the remaining available bandwidth resources, eight kilobits of excess information rate is allocated to each session based upon the priority of each session, until either all demands have been satisfied, or bandwidth resources are completely consumed.

When a guaranteed information rate cannot be provided to a requesting flow due to insufficient bandwidth resources, an admissions policy is invoked. The admissions policy may refuse a connection, display an error message or a combination of both. Admissions policies prevent service 'brown outs' which would otherwise occur if guaranteed information rate demand exceeds available bandwidth resources. By contrast, excess information rate requirements are not guaranteed. If a flow's request for excess information rate is not able to be satisfied from the available bandwidth resources, it is simply not met. In the previous example, if insufficient bandwidth existed in order to accommodate the ten voice over IP sessions, an admissions policy would be invoked.

Excess information rate (EIR) is redistributed each time a new flow is created, an existing flow is terminated, or a substantial change in demand for unreserved priority level occurs. By contrast, Guaranteed information rate (GIR) is

allocated only at the time of creation of a flow and freed when the flow terminates.

In a preferable embodiment, reserved TCP service levels may be enforced by TCP Rate Control for both inbound and outbound traffic. TCP Rate Control is a technique well known to persons of ordinary skill in the art which scales reserved flows as excess rate is available without incurring retransmission.

2.3 Isolating Bandwidth Resources for Traffic Classes

Bandwidth resources may be dedicated to certain traffic classes. There are two degrees of isolation. Hard isolation creates an entirely separate logical channel for designated traffic classes. For example, department A and department B could both be hard isolated, setting up two logical 0.75 Mbps channels on a T1 access link, one for each department. With hard isolation, unused bandwidth in either channel would not be shared with the other department's traffic.

Hard isolation involves allocating arbitrary subsets of access link bandwidth for any traffic class or family of traffic classes. It completely protects the traffic belonging to the respective class or classes, but this protection comes at the cost of forsaken opportunities to share unused bandwidth.

Soft isolation allows sharing of unused bandwidth, but limits the amount of guaranteed bandwidth that can be allocated to any particular traffic class at any one time. This provides a basic level of fairness between traffic classes which have corresponding guaranteed service policies. But this fairness does not come at the cost of efficiency gains accrued from sharing available bandwidth.

3.0 System Function

3.1 Building a Traffic Classification Tree

FIG. 2C depicts a flowchart **203** showing the process steps in building a classification tree, such as classification tree **201** of FIG. 2A, by adding traffic class nodes, such as traffic class node **206**, in order of specificity of classes. In a step **240**, a new tclass node is allocated for the traffic class which is to be added to the classification tree. Next, in a step **242**, a traffic specification is inserted for the new traffic class node, allocated in step **240**. Then, in a step **244**, the new tclass node is inserted into the classification tree. Next, in a step **246**, bandwidth resources allocated based upon the traffic class hierarchy is redistributed to reflect the new hierarchy.

FIG. 2D depicts a flowchart **205** showing the component steps of traffic specification creation step **242** of FIG. 2C. The processing steps of flowchart **205** insert a traffic specification, or tspec, into the traffic class allocated in step **240**. In a step **250**, a current tspec is initialized to the first tspec in the tclass. Next, in a decisional step **252**, a determination is made whether the new tspec is less specific than the current tspec, initialized in step **250**. If this is not the case, then in a step **258**, the new tspec is inserted after the current tspec, and processing returns. Otherwise, in a decisional step **254**, a determination is made whether this is the final tspec in the traffic class. If this is so, then in step **258**, the new tspec is inserted after the last (current) tspec. Otherwise, in a step **256**, the next tspec in the traffic class is examined by repeating processing steps **252**, **254**, **256** and **258**.

FIG. 2E depicts a flowchart **207** showing the component steps of traffic specification creation step **244** of FIG. 2D. The processing steps of flowchart **207** insert the traffic class, or tclass, allocated in step **240** into the classification tree. In a step **260**, a current sibling is initialized to the first sibling in the parent tclass. Next, in a decisional step **262**, a determination is made whether the new tclass is less specific than the current sibling, initialized in step **260**. If this is not the case, then in a step **268**, the new tclass is inserted after the current sibling, and processing returns. Otherwise, in a decisional step **264**, a determination is made whether this is the final sibling in the parent tclass. If this is so, then in step **268**, the new tclass is inserted after the last (current) tclass. Otherwise, in a step **266**, the next sibling in the parent traffic class is examined by repeating processing steps **262**, **264**, **266** and **268**.

3.2 Allocating Bandwidth based upon Policy

FIG. **3** depicts a functional block diagram **301** of component processes in accordance with a particular embodiment of the present invention. Processing of a new flow, such as new flow **300** as described by a flowchart **501** in FIG. **5A**, begins with a TCP autobaud component **302**, which implements the automatic detection of flow speeds as depicted in FIG. **1E**. TCP autobaud component **302** determines values for selectable information such as flow data rate of new flow **300**. A classifier **304** determines the class for the new flow using the information determined by TCP autobaud **302**. Having classified the traffic, the classifier next determines an appropriate policy for this particular flow from a classification tree (not shown). Based upon the policy, the classifier then determines a service level for the new flow. A bandwidth manager **306** uses the policy determined by classifier **304** in order to allocate bandwidth according to the service level prescribed by the policy.

FIG. **4A** depicts data structures in a particular embodiment according to the invention. The bandwidth management aspect of the present invention is performed between inside clients and servers and the access link. Outside clients and servers are on the other side of the access link. In the embodiment of FIG. **4A**, Traffic is directional. Inbound (flowing from an outside host to an inside host) traffic is managed entirely separately from outbound traffic.

A connection hash table **402** facilitates indexing into a plurality of TCP connections, each TCP connection having an associated flow, such as flows **300**, **299** and **298** of FIG. **3**. A plurality of TCP control blocks **404** are used to track connection-related information for each TCP connection. Host-related information is tracked using a plurality of host information control blocks **406**, indexed by a host hash table 408. Information about particular hosts corresponding to TCP connections is tracked using the TCP control block **404**.

Each TCP connection represented by TCP control block **404** has associated with it a gear **410** representing an outbound flow and a gear **411** representing an inbound flow. Each gear is an internal data object associated with a particular TCP connection or UDP session. Gears are protocol independent and contain rate-based information. Since network resources are managed separately for inbound and outbound traffic in this particular embodiment, there is a bandwidth pool **412** for outbound traffic and a bandwidth pool **413** for inbound traffic. Bandwidth pools represent collections of available network resources managed by bandwidth manager **306**. Bandwidth pools are arbitrarily divided into a partition **414** in the outbound path, and a partition **415** in the inbound path. Each partition represents an arbitrarily-sized portion of available network resource. Gears become associated with particular partitions during the beginning of flow management for the new flow in step

518 of FIG. **5A**. A policy **416** in the outbound path and a policy **417** in the inbound path become associated with partition **414** in the outbound path and partition **415** in the inbound path during the beginning of management of a new flow, as shown in FIG. **5A**. These policies are associated with particular flows according to the traffic class that classifier **304** assigns to the particular flow.

FIG. **5A** depicts flowchart **501** of the steps taken in initiating a new flow in accordance with a particular embodiment of the invention. In a step **502**, TCP autobaud component **302** detects a TCP data rate of new flow **300**. In a step **504**, classifier **304** classifies the new flow to determine a policy. Classification is described in greater detail in FIG. **5F** herein below. In a step **506**, based upon the policy determined in step **504**, a guaranteed information rate (GIR) and an excess information rate (EIR) are determined and speed scaled to the incoming TCP data rate, detected in step **502**. Speed scaling is described in greater detail in FIGS. **5G** and **5H** herein below. In a step **508**, bandwidth manager **306** allocates guaranteed information rate resources and excess information rate resources to accommodate an initial target rate for the new flow. In a decisional step **510**, bandwidth manager **306** determines if a partition limit would be exceeded by the allocation determined in step **508**. If this is the case, then in a step **512**, an admissions policy is invoked. Otherwise, in a step **514**, the excess information rate resources demanded by new flow **300** is added to a total excess information rate resources demanded for this bandwidth pool. In a step **516**, excess information rate resources are redistributed with respect to all flows within the partition, if demand has changed substantially since the last time it was redistributed. FIG. **5E** depicts a flowchart **509** showing the steps in reapplying demand satisfaction parameters to all traffic flows in the bandwidth pool. A new target rate and total excess information rate resources are calculated. Finally, in a step **518**, the new flow is associated with a partition.

FIG. **5B** depicts a flowchart **503** showing processing of a packet belonging to an existing flow such as a forward flow packet **299**. In a decisional step **520**, a scheduler **308**, using policy information determined at flow initiation, filters TCP acknowledgments for scheduling in a step **522**. Output times are scheduled on millisecond boundaries, based upon the target data rate computed in step **508** of FIG. **5A**. In as step **524**, the remaining packets are delivered immediately to TCP, which performs just in time delivery.

FIG. **5C** depicts a flowchart **505** showing processing by bandwidth manager **306** of a packet at the termination of a flow, such as an end flow packet **298**. In a step **530**, guaranteed information rate resources are restored. In step **532**, excess information rate resources demanded by the terminating flow **298** is subtracted from the total excess information rate resources demanded. In a step **534**, excess information rate resources are redistributed as described in FIG. **5D**. Finally, in a step **536**, the flow is removed from its partition.

FIG. **4B** depicts specific data structures used in redistributing unreserved service resources among multiple flows in a particular partition. Input rates of particular flows, such as new flow **300**, are measured by TCP autobaud component **302** and used to determine an Extrapolated Unreserved Demand or (EUD) **424**. Flows requiring excess information rate resources, such as flows **299** and **298**, having gears **410** and **411** (not shown), respectively, have associated with them an individual flow demands **420** and **421**. The individual flow demands of all flows in a particular partition are added together to form an Aggregate Rate Demand (ARD)

422 at each priority level. Aggregate rate demand is added to extrapolated unreserved demand 424 to arrive at a total instantaneous demand 426. The total instantaneous demand is satisfied from the highest priority, priority 7, downward based upon the available excess information rate resources in the partition 414. This available rate is information rate which has not been used to satisfy guaranteed information rate demands. From the available excess information rate resources, a satisfaction percentage vector 428 is computed having the percentage of each priority of flow which may be satisfied in the total demand vector 426 by the available excess information rate resources in partition 414. Individual flow demand vectors 420 and 421 are each multiplied by the satisfaction percentage for each priority 428 in order to arrive at a target rate which is incorporated in gears 410 and 411 corresponding to each flow. The satisfaction percentage vector 428 is also applied to the Extrapolated Unreserved Demand (EUD) 424 to determine unreserved targets.

3.3 Reconciling Needs for Bandwidth

FIG. 5D depicts flowchart 507 of the processing steps to redistribute excess information rate resources with respect to reserved service flows within a given partition. In a step 540, a demand satisfaction metric is computed for the bandwidth pool being redistributed. In a decisional step 542, a determination whether the demand satisfaction metric computed in step 540 has changed substantially since the last time demand was redistributed. If this is the case, then in a decisional step 544, a loop is entered for each gear associated with the partition owning the bandwidth pool being redistributed. In a step 546, demand satisfaction parameters are reapplied a traffic flow associated with each gear and a new target rate and a new total excess information rate resources are calculated. Processing continues until in decisional step 544, it is determined that all gears have been processed.

FIG. 5E depicts flowchart 509 of the processing steps for reconciling bandwidth among a plurality of flows, requiring reserved service and unreserved service resources. For example, partition 414 of FIG. 4B has unreserved service flow 300 and reserved service flows 299 and 298. In a decisional step 550, a loop is formed with a step 554, in which the individual flow demands of reserved service flows, such as flows 299 and 298 in partition 414, are added together to form an aggregate rate demand (ARD) 422. Next, in a step 552, an extrapolated unreserved demand (EUD) 424 is determined from the input rate of unreserved service traffic 300 in partition 414. In a preferable embodiment, the EUD must be inflated in order to promote growth in traffic. This is done to accommodate the elastic nature of TCP traffic. Next, in a step 556, the EUD determined in step 552 and the ARD determined in step 554 are combined to form an instantaneous total demand (TD) 426 for the entire partition 414. The total instantaneous demand is satisfied from the highest priority level, level 7, downward based upon the available unreserved resources in the partition 414. In a step 558, a satisfaction percentage vector (SP) 428 is computed for the partition from the total demand vector (TD) 426 determined in step 556. The satisfaction percentage indicates that percentage of the demand at each priority level that can be satisfied from the unreserved resources in the partition. Next, in a step 560, a target rate for reserved rate based flows, such as flows 299 and 298, is computed. Finally, in a step 562, a target rate for unreserved priority based flows, such as flow 300, is computed.

3.4 Searching a Classification Tree

FIG. 5F depicts a flowchart 511 showing the component steps of traffic classification step 504 of FIG. 5A. The processing steps of flowchart 511 determine a class and a policy for a flow, such as new flow 300, by traversing a classification tree such as the classification tree 201 in FIG. 2A. Processing begins with a first node in the classification tree 201, such as department A node 202. In a step 570, a traffic specification of a child node, such as FTP node 206 is compared with a flow specification of the new flow 300. In a decisional step 572, if a match is discovered, then in a step 574, the processing of flowchart 511 is applied to the child node 206 recursively. Otherwise, if child node 206 does not match the new flow, then in a decisional step 576, processing determines whether any sibling nodes exist for the child node 206. If processing detects the existence of a sibling of child node 206, such as child node 208, then processing continues on node 208 with step 570. Otherwise, if there are no further child nodes for node 202, then in a decisional step 578, a determination is made whether the node is a leaf node having a policy. If no policy exists, then in a step 580, processing backtracks to a parent node and looks for a policy associated with the parent node to apply to the new flow 300. Otherwise, if a policy is associated with the node 202, then in a step 582, the policy is associated with the new flow 300.

3.5 Speed Scaling

FIG. 5G depicts a flowchart 513 showing the component steps of speed scaling step 506 of FIG. 5A. The processing steps of flowchart 513 determine an acceptable allocation of bandwidth to reserved service flows, i.e., GIR and EIR, by allocating rate for each gear, such as gears 410 and 411, based upon individual flow demands, base limits and total limits. In a step 582, scaled rate for GIR is set based upon a connection speed detected by TCP autobaud component 302 for a particular flow, such as flow 299, having a gear, such as gear 410. Then in a step 584, scaled rate for EIR is set based upon a connection speed detected by TCP autobaud component 302 for a particular flow, such as flow 298, having a gear, such as gear 411. Next, in a step 586, a limit is computed from either the available EIR remaining after the allocation to flow 299 in step 584, or from a total limit, if such a limit is specified. Then, in a step 588, a determination is made whether any extra EIR resource remains after the limit computed in step 586 has been satisfied. Then, in a step 590, limits computed in step 586 are allocated to flows, such as flow 299 and flow 298, in order of descending priority level. In a decisional step 592, the processing of steps 582 through 592 are applied to subsequent gears, if any.

FIG. 5H depicts a flowchart 515 showing the component steps of scaled rate setting steps 582 and 584 of FIG. 5G. The processing steps of flowchart 515 determine whether the rate of the underlying link detected by TCP autobaud 302 exceeds limits imposed on the network. In a decisional step 592, a determination is made whether the detected speed of a flow is less than a base limit minimum. If this is so, then in a step 594, the base limit is returned as the scaled rate. Otherwise, in a decisional step 596, a determination is made whether the total rate of all flows is greater than a maximum limit. If this is so, then in a step 598, the maximum limit is returned as the scaled rate. Otherwise, in a decisional step 600, a determination is made whether the detected speed of a flow is greater than a total rate of all flows. If this is so, then in a step 602, the maximum limit is returned as the scaled rate. Otherwise, in a step 604, a percentage of the total rate, represented by the maximum limit, is computed. Then, in a step 606, the percentage computed in step 604 is applied to a total rate available, i.e., the maximum limit—base (minimum) limit. The result is returned as the scaled rate.

4.0 Conclusion

In conclusion, the present invention provides for a policy based bandwidth allocation method for IP packet telecommunications systems wherein bandwidth is allocated to requesting flows according to automatically determined application requirements. An advantage of network management techniques according to the present invention is that network managers need only define traffic classes which are of interest. A further advantage of the present invention is that traffic classes may include information such as a URI for web traffic. A yet further advantage of the present invention is that service levels may be defined in terms of explicit rates, may be scaled to a remote client or server's network access rate, specified for high speed and low speed users and defined in terms of a guaranteed minimum service level.

Other embodiments of the present invention and its individual components will become readily apparent to those skilled in the art from the foregoing detailed description. As will be realized, the invention is capable of other and different embodiments, and its several details are capable of modifications in various obvious respects, all without departing from the spirit and the scope of the present invention. Accordingly, the drawings and detailed description are to be regarded as illustrative in nature and not as restrictive. It is therefore not intended that the invention be limited except as indicated by the appended claims.

What is claimed is:

1. A method for allocating bandwidth in a connection-less network system having an arbitrary number of flows of packets, including zero, using a classification paradigm, said allocation method comprising the steps of:

parsing a packet into a flow specification, wherein said flow specification contains at least one instance of any of the following:
a protocol family designation,
a direction of packet flow designation,
a protocol type designation,
a plurality of hosts,
a plurality of ports,
in http protocol packets, a pointer to a URL; thereupon,

matching the flow specification of the parsing step to a plurality of hierarchically-recognized classes represented by a plurality of nodes, each node having a traffic specification and a mask, according to the mask; thereupon,

having found a matching node in the matching step, associating said flow specification with one class of said plurality of hierarchically-recognized classes represented by a plurality nodes; and

allocating bandwidth resources according to a policy associated with said class.

2. The method of claim 1 wherein selected ones of said nodes include at least one traffic specifications.

3. The method of claim 1 wherein said matching step includes matching said plurality of hierarchically-recognized classes as a plurality of orthogonal flow attributes.

4. The method of claim 1 wherein said step of allocating bandwidth resources further comprises the steps of:

for rate based policies having a guaranteed information rate (GIR) and an excess information rate (EIR), allocating GIR until no further GIR is available; thereupon, allocating EIR,

for priority based policies, having a priority, allocating unreserved service according to said priority.

5. The method of claim 1 wherein said step of allocating bandwidth resources further comprises the step of:

allocating a combination of GIR resources, EIR resources and unreserved resources, wherein GIR resources and EIR resources are allocated based upon speed scaling and unreserved resources are allocated based upon a priority.

6. In a packet communication environment allocated into layers, including at least a transport layer, link layer and an application layer and having defined traffic classes, a method for classifying Internet Protocol (IP) flows at the transport layer for use in determining a policy, or rule of assignment of a service level, and enforcing that policy by direct rate control, said method comprising:

applying individual instances of traffic classes to a classification tree for said defined traffic classes based on selectable information obtained from an application layer of a communication protocol; and thereupon,

mapping the IP flow to the defined traffic classes, said defined traffic classes having an associated policy.

7. A method for managing bandwidth on Internet Protocol (IP) flows in a packet communication environment allocated into layers, including at least a transport layer, a link layer and an application layer, said method comprising:

automatically detecting selectable information about each one of said flows;

determining a policy for assigning a service level to said flows based upon said selectable information automatically detected about one of said flows; and

implementing said policy by explicit data rate control of said one of said flows.

8. The method of claim 7 wherein said determining step further comprises applying individual instances of traffic class specifications, each traffic class specification having an associated policy, to said one of said flows based on said selectable information; thereupon,

associating with said one of said flows the particular policy associated with said traffic class.

9. The method of claim 8 wherein each said traffic class has a hierarchical relationship to one another.

10. A method for allocating network bandwidth, which bandwidth comprises guaranteed information rate (GIR) resources, excess information rate (EIR) resources and unreserved rate resources, on Internet Protocol (IP) flows of packets in a connection-less network service environment allocated into layers, including at least a transport layer, a link layer and an application layer, said method comprising:

automatically detecting selectable information about one of said flows;

determining a policy for assigning a service level to said one of said flows based upon said automatically detected selectable information about said one of said flows, said policy directing either reserved service or unreserved service; and

dynamically allocating said bandwidth representing a combination of GIR resources, EIR resources and unreserved resources among a plurality of said flows assigned to said service level by said determining step.

11. A method for allocating network bandwidth, which bandwidth comprises guaranteed information rate (GIR) resources, excess information rate (EIR) resources and unreserved rate resources, on Internet Protocol (IP) flows of packets in a connection-less network service environment allocated into layers, including at least a transport layer, a link layer and an application layer, said method comprising:

automatically detecting selectable information about one of said flows;

determining a policy for assigning a service level to said one of said flows based upon said automatically detected selectable information about said one of said flows, said policy directing either reserved service or unreserved service; and

dynamically allocating said bandwidth representing a combination of GIR resources, EIR resources and unreserved resources among a plurality of said flows assigned to said service level by said determining step;

wherein said unreserved resources comprise a plurality of individual inputs, each input having a priority, further comprising the step of:

extrapolating from input rate an extrapolated unreserved demand (EUD);

determining a target rate for each said input according to said priority by multiplying said extrapolated unreserved demand by said satisfaction percentage of demand determined in the allocating step.

12. The method of claim **10** wherein said step of allocating bandwith representing a combination of GIR resources, EIR resources and unreserved resources comprises allocating GIR resources and EIR resources based upon a speed scaling and allocating unreserved resources based upon a priority.

13. A system for allocating bandwidth in a connectionless network service environment having any number of flows of flows, including zero, using a classification paradigm comprising:

a network routing device;

means for classifying a plurality of flows;

means for allocating bandwidth; and

means for enforcing said bandwidth allocation.

14. The system according to claim **13** wherein said network routing device further comprises:

a processor means; and

a plurality of network interfaces.

15. The system according to claim **13** wherein said classifying means includes means for stepwise mapping of policies according to association with nodes classified in a hierarchy.

16. The method according to claim **1** wherein said matching step comprises stepwise mapping of policies according to association with nodes classified in a hierarchy.

17. The method according to claim **10**, further including steps of stepwise mapping of policies according to association with nodes classified in a hierarchy.

* * * * *

US006519636B2

(12) **United States Patent**
Engel et al.

(10) **Patent No.: US 6,519,636 B2**
(45) **Date of Patent: *Feb. 11, 2003**

(54) EFFICIENT CLASSIFICATION, MANIPULATION, AND CONTROL OF NETWORK TRANSMISSIONS BY ASSOCIATING NETWORK FLOWS WITH RULE BASED FUNCTIONS

(75) Inventors: **Robert Engel**, Greenwich, CT (US); **Tsipora P. Barzilai**, Tel Aviv (IL); **Dilip Dinkar Kandlur**, Yorktown Heights, NY (US); **Ashish Mehra**, White Plains, NY (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/181,376**

(22) Filed: **Oct. 28, 1998**

(65) **Prior Publication Data**

US 2003/0005144 A1 Jan. 2, 2003

(51) **Int. Cl.**^{7} **G06F 15/173**

(52) **U.S. Cl.** **709/223**; 709/203; 709/224; 709/231; 709/232; 709/233; 709/250; 370/229; 370/252

(58) **Field of Search** 709/200–203, 709/217–219, 223–225, 230–233, 236–239, 249–250; 713/200–201; 707/10, 101–104; 370/224–232, 252, 473

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,425,029 A	* 6/1995	Hluchyj et al.	....	370/235
5,568,487 A	* 10/1996	Sitbon et al.	....	709/230
5,682,534 A	* 10/1997	Kapoor et al.	....	709/203
5,727,002 A	* 3/1998	Miller et al.	....	709/237
5,822,524 A	* 10/1998	Chen et al.	....	709/203
5,854,841 A	* 12/1998	Nakata et al.	....	709/203
5,889,954 A	* 3/1999	Gessel et al.	....	709/223
5,903,724 A	* 5/1999	Takamoto et al.	....	709/200
6,021,440 A	* 2/2000	Post et al.	....	709/231
6,046,979 A	* 4/2000	Bauman	....	370/229
6,061,796 A	* 5/2000	Chen et al.	....	709/225
6,269,467 B1	* 7/2001	Chang et al.	....	716/1

OTHER PUBLICATIONS

Microsoft Press, "Computer Dictionary" (1997), pp. 207 and 348.*

Microsoft Press, "Computer Dictionary" (1997), pp. 114 and 466.*

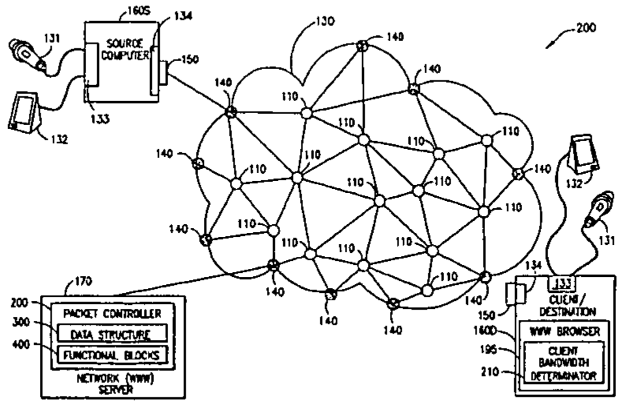
* cited by examiner

Primary Examiner—Bharat Barot
(74) *Attorney, Agent, or Firm*—Louis J. Percello; Perman & Green, LLP

(57) **ABSTRACT**

A computer connected to one or more networks through appropriate network interfaces is used to classify, manipulate, and/or control communications, e.g., packets sent and/or received over the network by one or more applications executing in the computer. Each application is connected to the network through one or more sockets to enable this communication. The computer also comprises one or more rule sets of one or more rules. A socket set of one or more of the sockets is associated with only one of the rule sets. The rules in the rule set are used to control one or more of the packets communicated by the applications communicating over the socket(s) associated with the respective rule set. Rules can be added to the rule set, deleted from the rule set, or modified in order to classify, manipulate, and/or control the communication of the packets, e.g. to control the rate at which the packets are sent or to provide certain security functions.

21 Claims, 11 Drawing Sheets



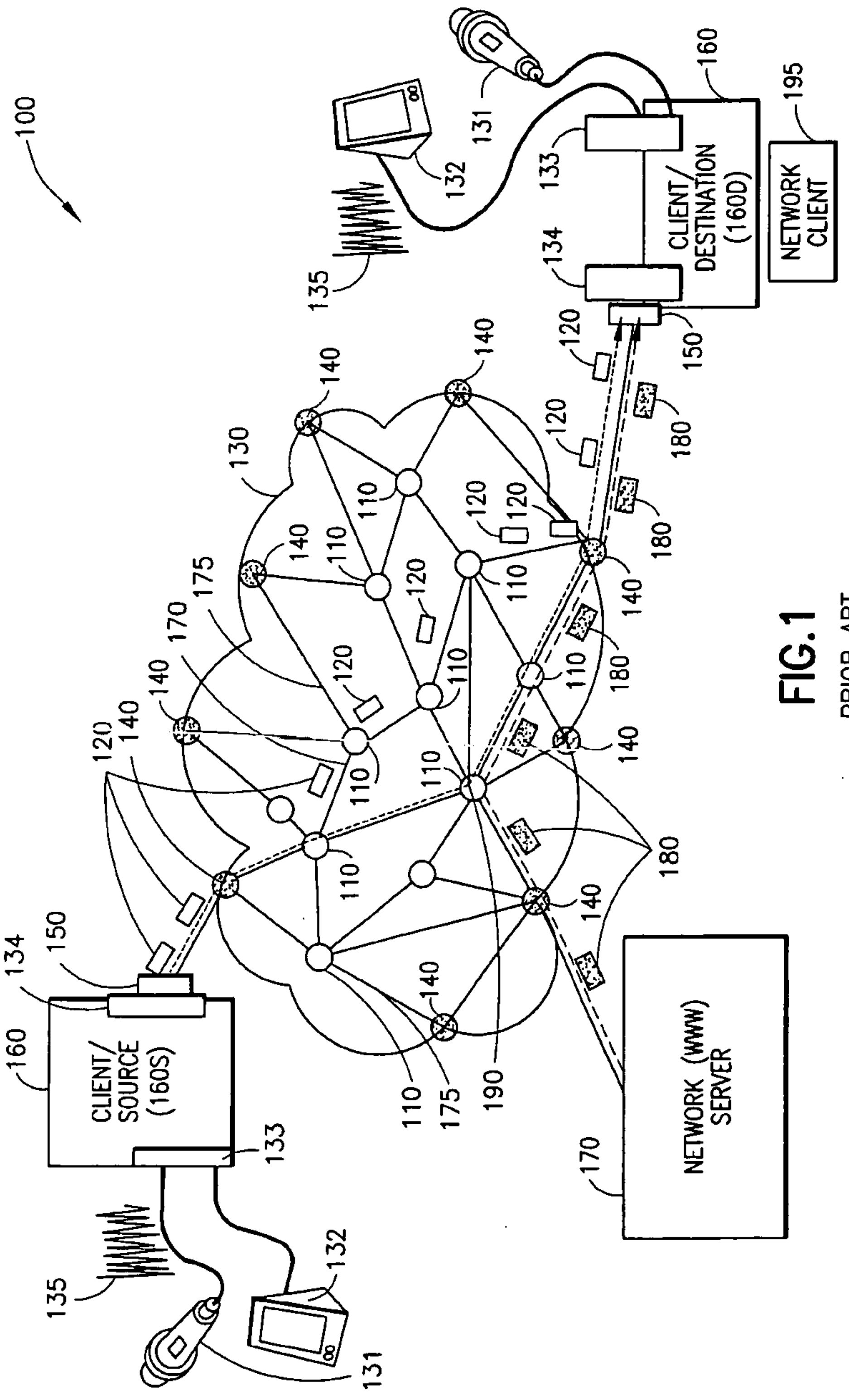


FIG. 1
PRIOR ART

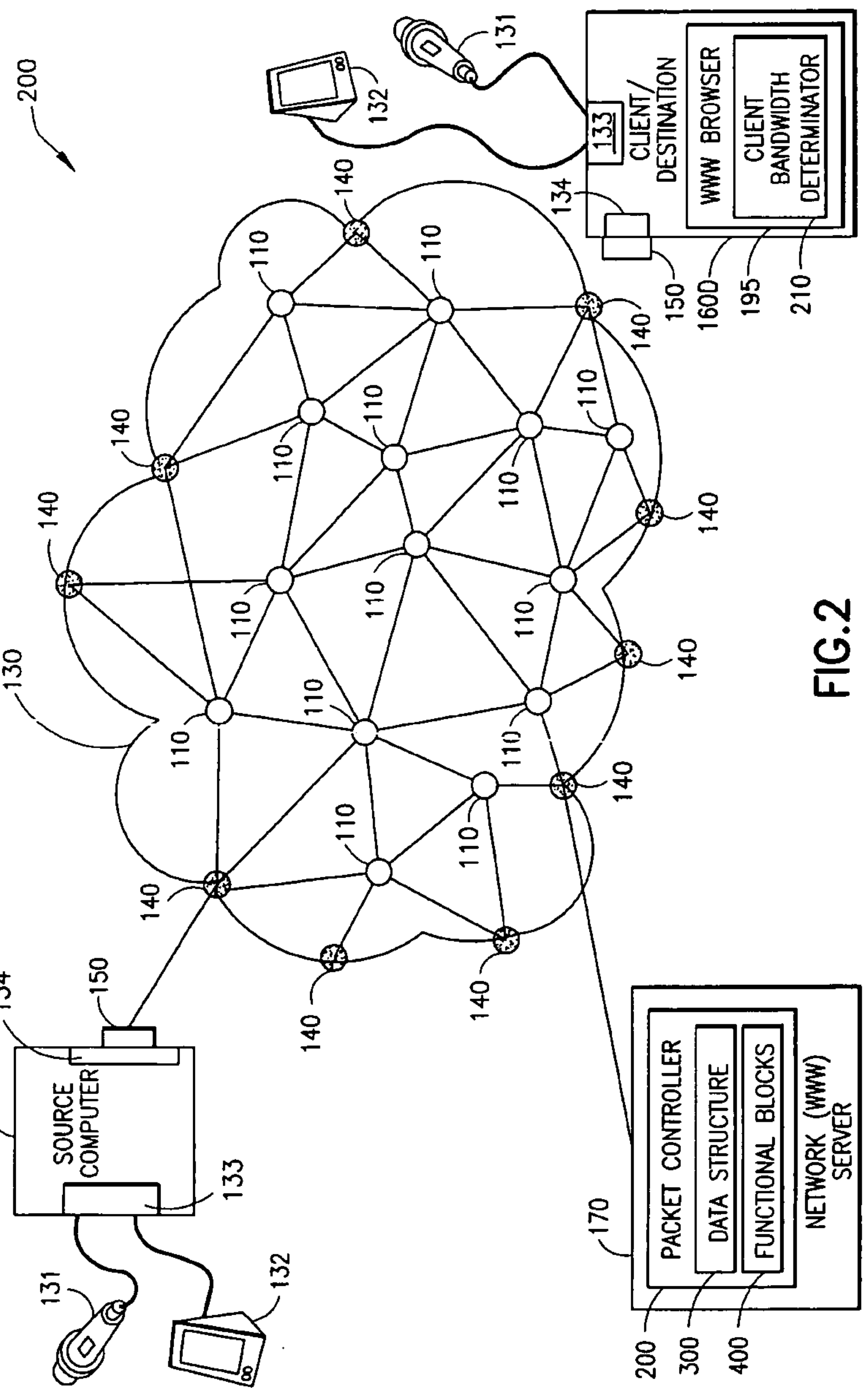

200
130
110
140
131
132
133
134
150
160D
195
210
CLIENT/ DESTINATION
WWW BROWSER
CLIENT BANDWIDTH DETERMINATOR
160S
SOURCE COMPUTER
170
PACKET CONTROLLER
DATA STRUCTURE
FUNCTIONAL BLOCKS
NETWORK (WWW) SERVER
300
400

FIG.2

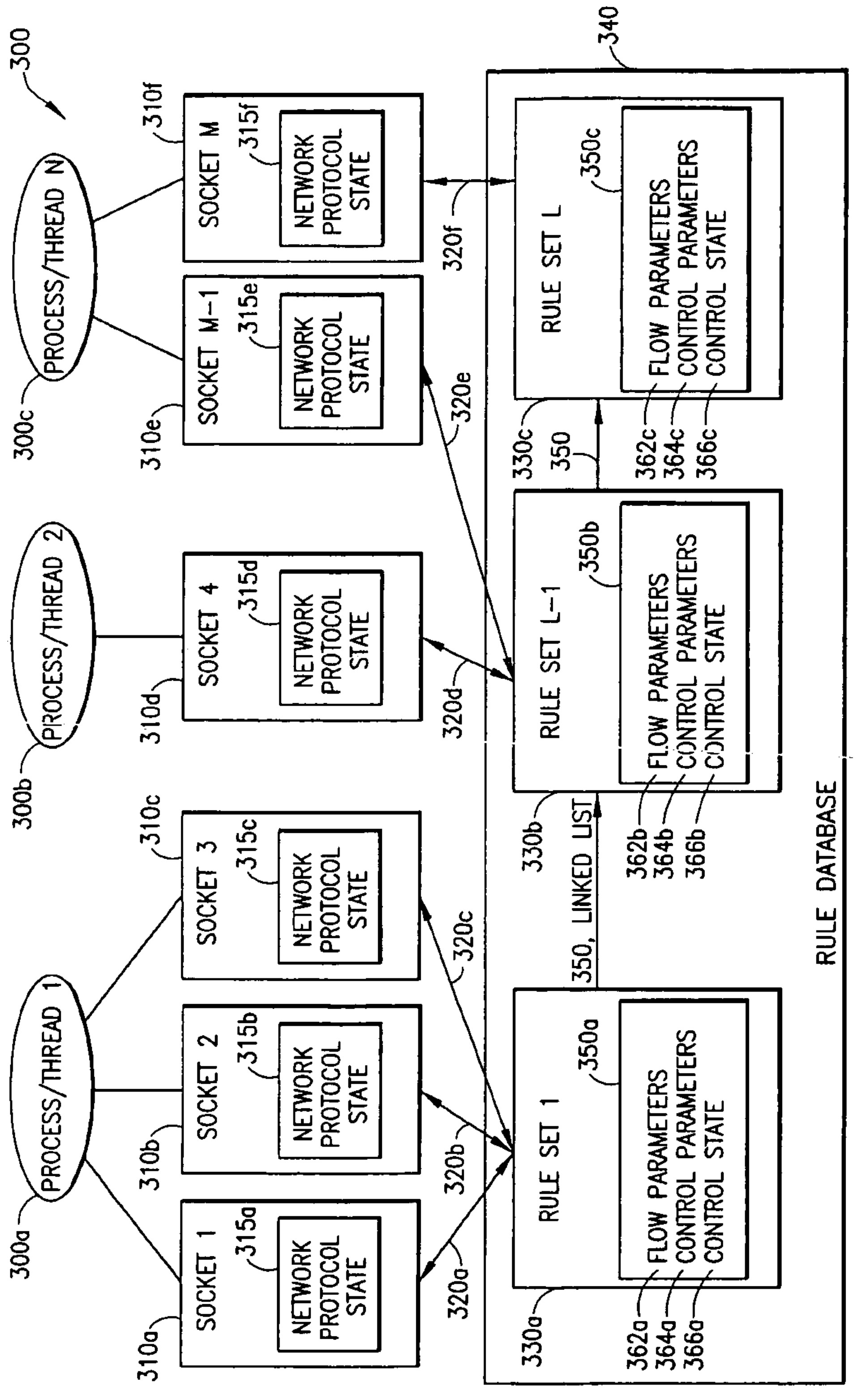

300
300a
PROCESS/THREAD 1
300b
PROCESS/THREAD 2
300c
PROCESS/THREAD N
310a
SOCKET 1
315a
NETWORK PROTOCOL STATE
310b
SOCKET 2
315b
NETWORK PROTOCOL STATE
310c
SOCKET 3
315c
NETWORK PROTOCOL STATE
310d
SOCKET 4
315d
NETWORK PROTOCOL STATE
310e
SOCKET M-1
315e
NETWORK PROTOCOL STATE
310f
SOCKET M
315f
NETWORK PROTOCOL STATE
320a
320b
320c
320d
320e
320f
340
330a
RULE SET 1
350a
FLOW PARAMETERS
CONTROL PARAMETERS
CONTROL STATE
362a
364a
366a
350, LINKED LIST
330b
RULE SET L-1
350b
FLOW PARAMETERS
CONTROL PARAMETERS
CONTROL STATE
362b
364b
366b
350
330c
RULE SET L
350c
FLOW PARAMETERS
CONTROL PARAMETERS
CONTROL STATE
362c
364c
366c
RULE DATABASE

FIG.3

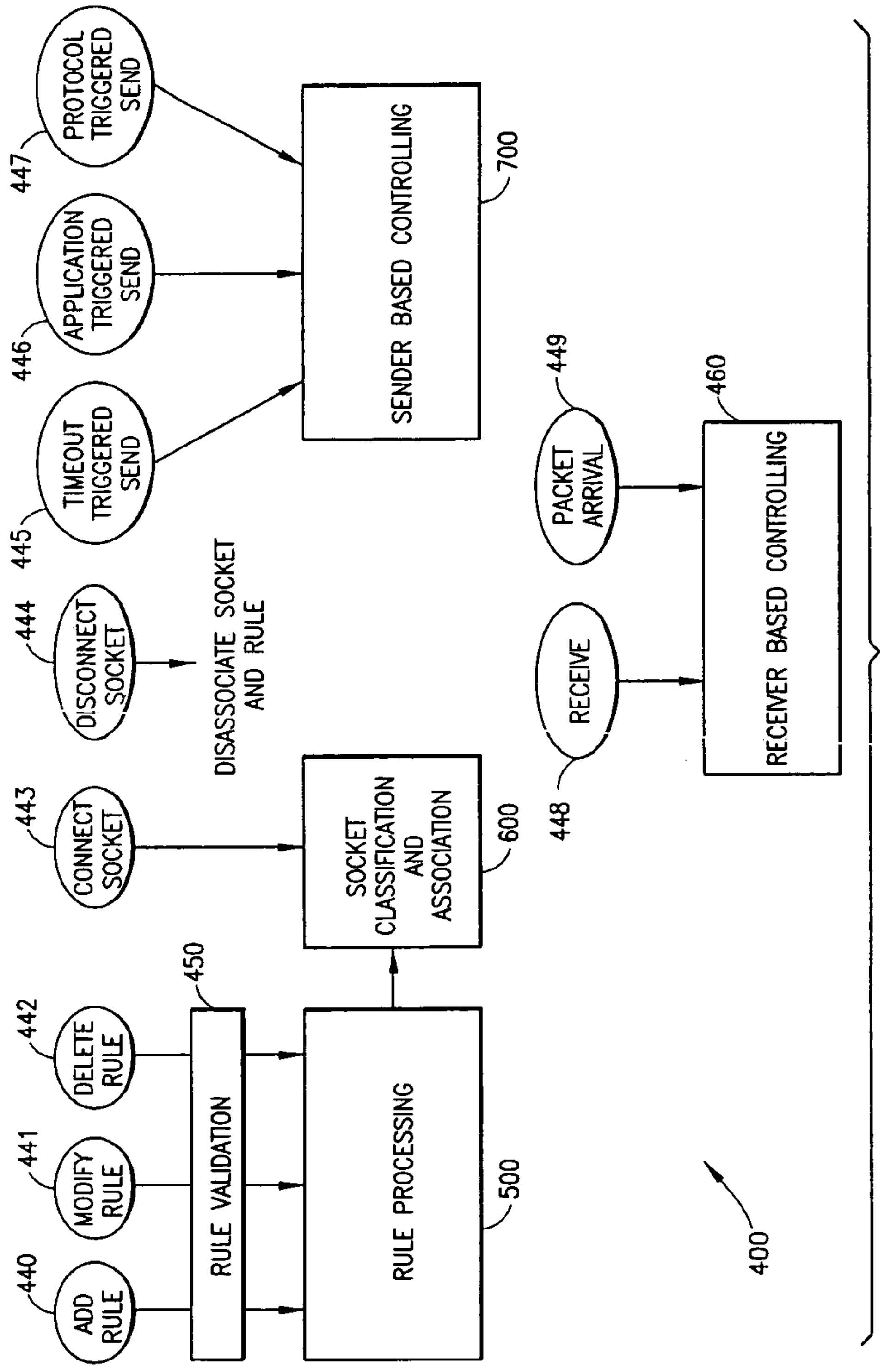

440 ADD RULE
441 MODIFY RULE
442 DELETE RULE
450 RULE VALIDATION
500 RULE PROCESSING
443 CONNECT SOCKET
600 SOCKET CLASSIFICATION AND ASSOCIATION
444 DISCONNECT SOCKET
DISASSOCIATE SOCKET AND RULE
445 TIMEOUT TRIGGERED SEND
446 APPLICATION TRIGGERED SEND
447 PROTOCOL TRIGGERED SEND
700 SENDER BASED CONTROLLING
448 RECEIVE
449 PACKET ARRIVAL
460 RECEIVER BASED CONTROLLING
400

FIG.4

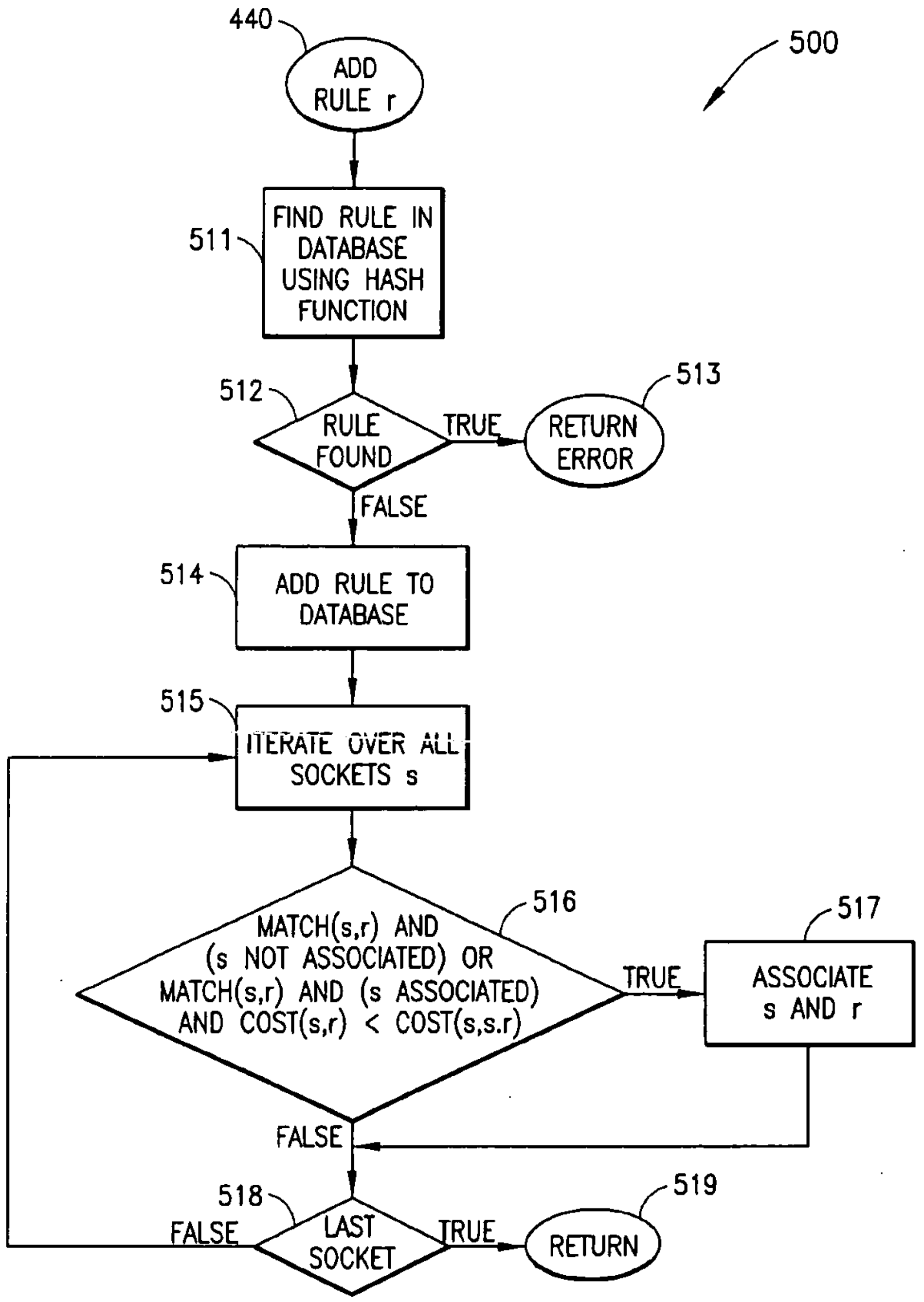

500
440
ADD RULE r
511
FIND RULE IN DATABASE USING HASH FUNCTION
512
RULE FOUND
TRUE
513
RETURN ERROR
FALSE
514
ADD RULE TO DATABASE
515
ITERATE OVER ALL SOCKETS s
516
MATCH(s,r) AND (s NOT ASSOCIATED) OR MATCH(s,r) AND (s ASSOCIATED) AND COST(s,r) < COST(s,s.r)
TRUE
517
ASSOCIATE s AND r
FALSE
518
LAST SOCKET
FALSE
TRUE
519
RETURN

FIG.5a

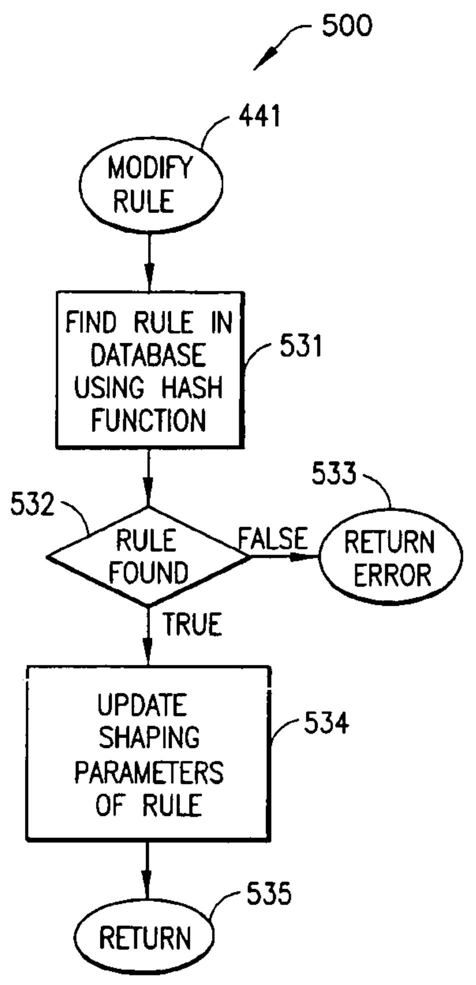

500
441
MODIFY RULE
FIND RULE IN DATABASE USING HASH FUNCTION
531
532
RULE FOUND
FALSE
533
RETURN ERROR
TRUE
UPDATE SHAPING PARAMETERS OF RULE
534
RETURN
535

FIG.5b

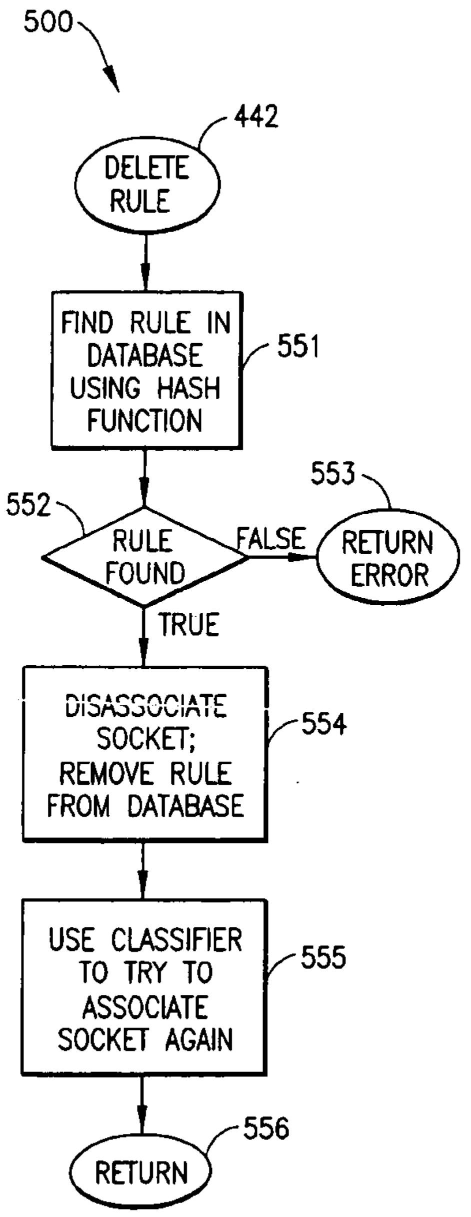
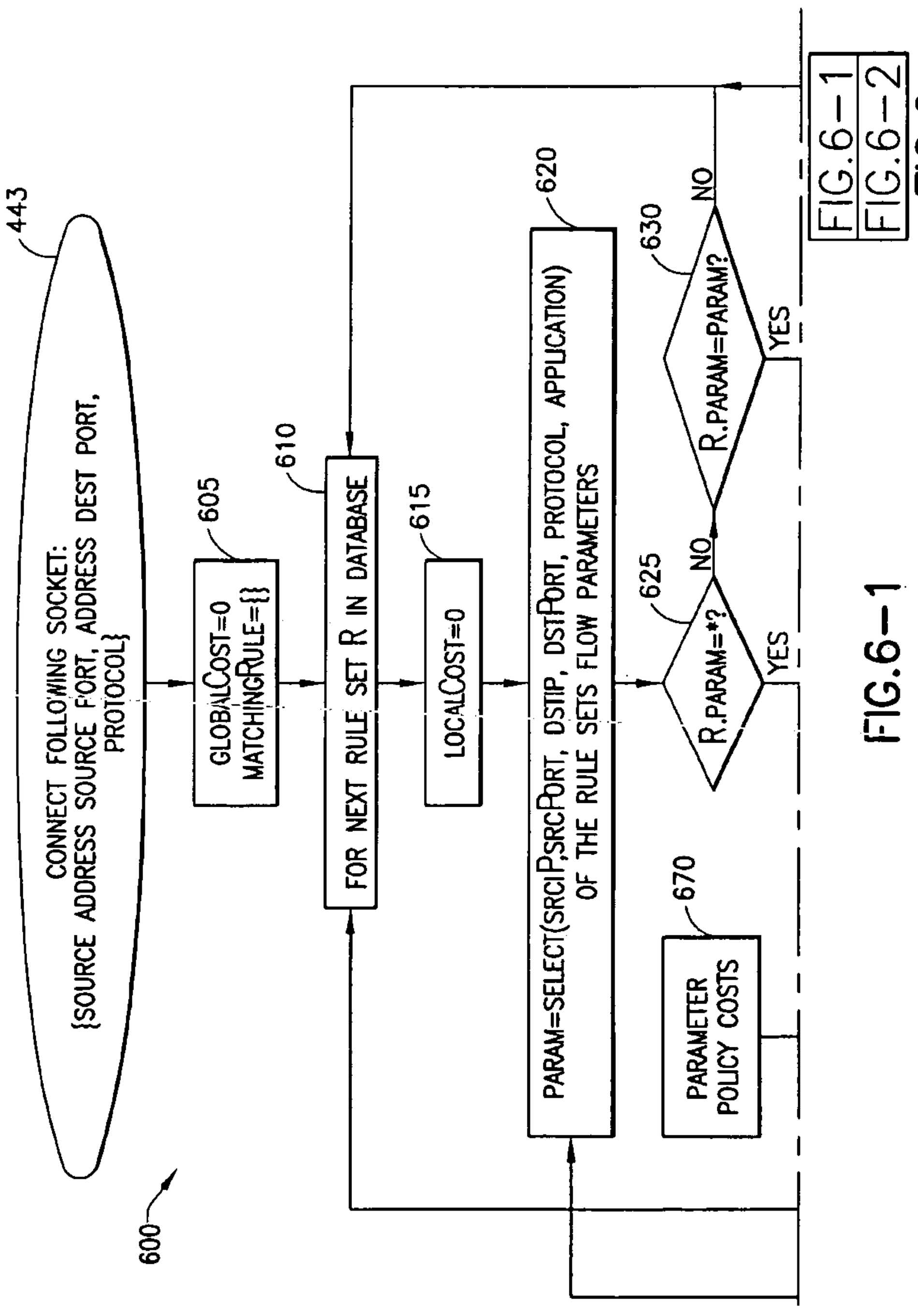

500
442
DELETE RULE
FIND RULE IN DATABASE USING HASH FUNCTION
551
552
RULE FOUND
FALSE
553
RETURN ERROR
TRUE
DISASSOCIATE SOCKET; REMOVE RULE FROM DATABASE
554
USE CLASSIFIER TO TRY TO ASSOCIATE SOCKET AGAIN
555
RETURN
556

FIG.5c



443
CONNECT FOLLOWING SOCKET: {SOURCE ADDRESS SOURCE PORT, ADDRESS DEST PORT, PROTOCOL}
605
GLOBALCOST=0 MATCHINGRULE={}
610
FOR NEXT RULE SET R IN DATABASE
615
LOCALCOST=0
620
PARAM=SELECT(SRCIP,SRCPORT, DSTIP, DSTPORT, PROTOCOL, APPLICATION) OF THE RULE SETS FLOW PARAMETERS
625
R.PARAM=*?
YES
NO
630
R.PARAM=PARAM?
YES
NO
670
PARAMETER POLICY COSTS
600
FIG.6-1
FIG.6-2
FIG.6

FIG.6-1

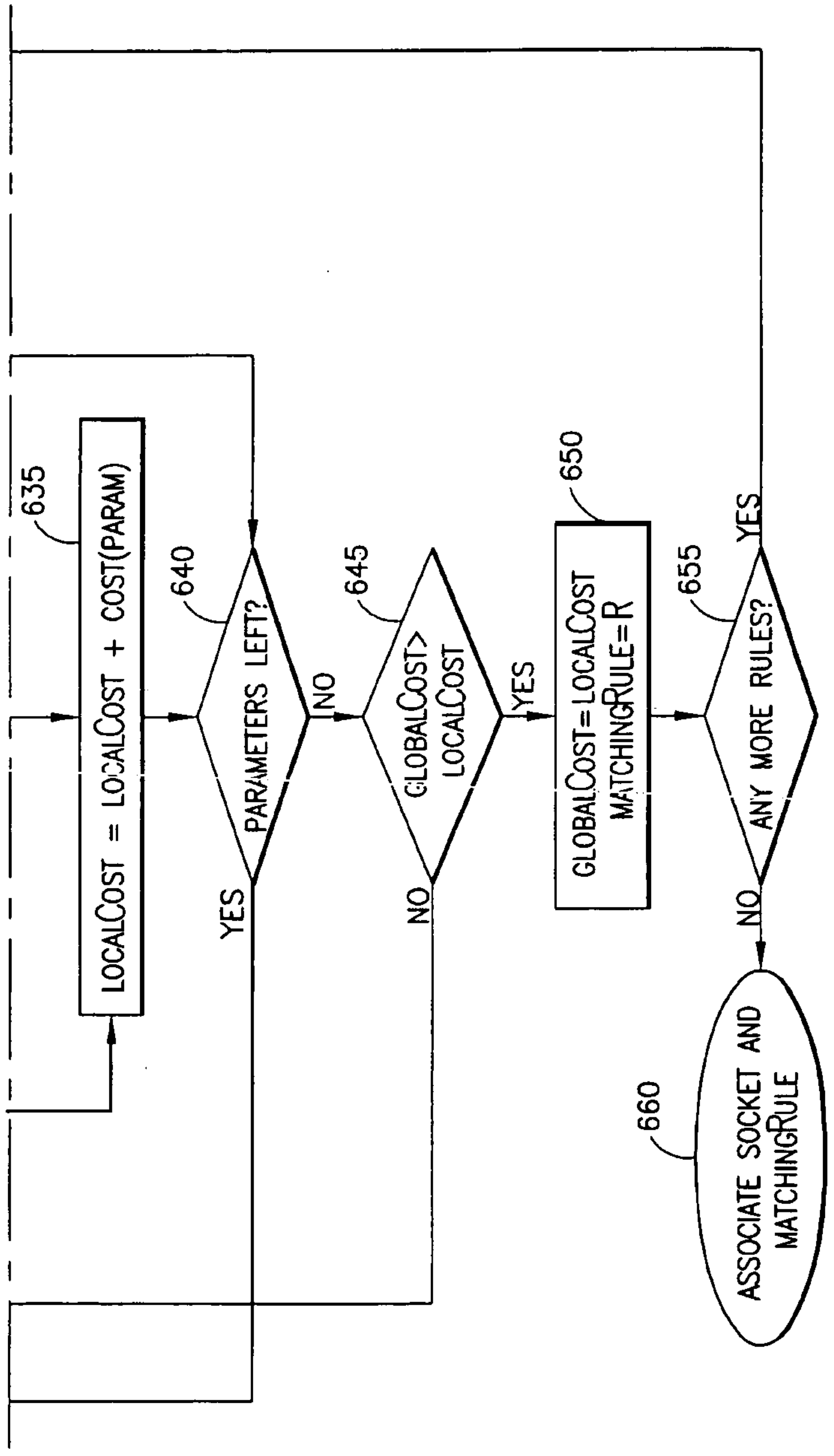

635
LOCALCOST = LOCALCOST + COST(PARAM)
640
PARAMETERS LEFT?
YES
NO
645
GLOBALCOST> LOCALCOST
YES
NO
650
GLOBALCOST=LOCALCOST
MATCHINGRULE=R
655
ANY MORE RULES?
YES
NO
660
ASSOCIATE SOCKET AND MATCHINGRULE

FIG.6–2

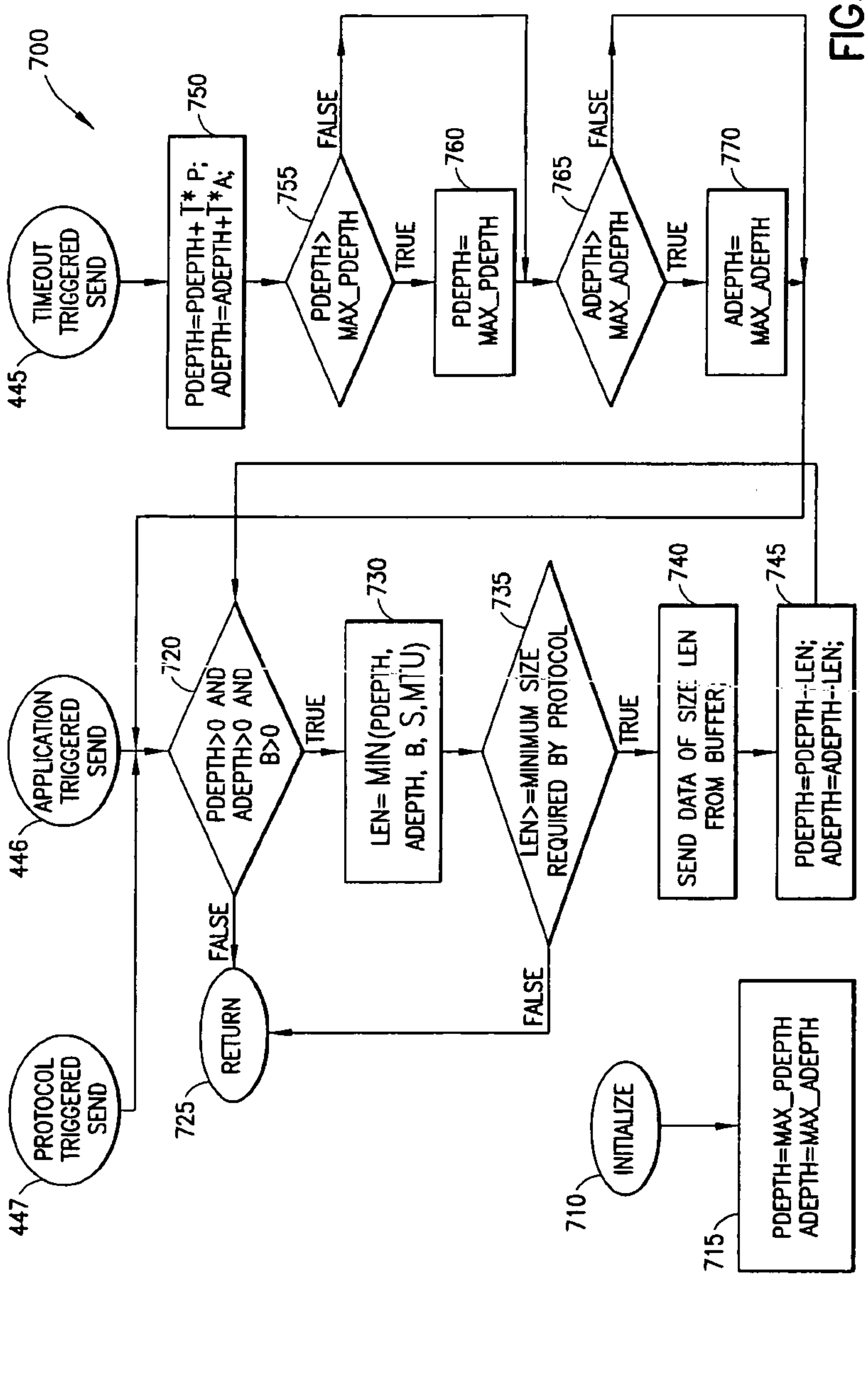

700
445
TIMEOUT TRIGGERED SEND
750
PDEPTH=PDEPTH+T* P;
ADEPTH=ADEPTH+T*A;
755
PDEPTH> MAX_PDEPTH
FALSE
TRUE
760
PDEPTH= MAX_PDEPTH
765
ADEPTH> MAX_ADEPTH
FALSE
TRUE
770
ADEPTH= MAX_ADEPTH
446
APPLICATION TRIGGERED SEND
720
PDEPTH>0 AND ADEPTH>0 AND B>0
TRUE
FALSE
730
LEN= MIN(PDEPTH, ADEPTH, B, S,MTU)
735
LEN>=MINIMUM SIZE REQUIRED BY PROTOCOL
TRUE
FALSE
740
SEND DATA OF SIZE LEN FROM BUFFER;
745
PDEPTH=PDEPTH-LEN;
ADEPTH=ADEPTH-LEN;
447
PROTOCOL TRIGGERED SEND
725
RETURN
710
INITIALIZE
715
PDEPTH=MAX_PDEPTH
ADEPTH=MAX_ADEPTH

FIG.7

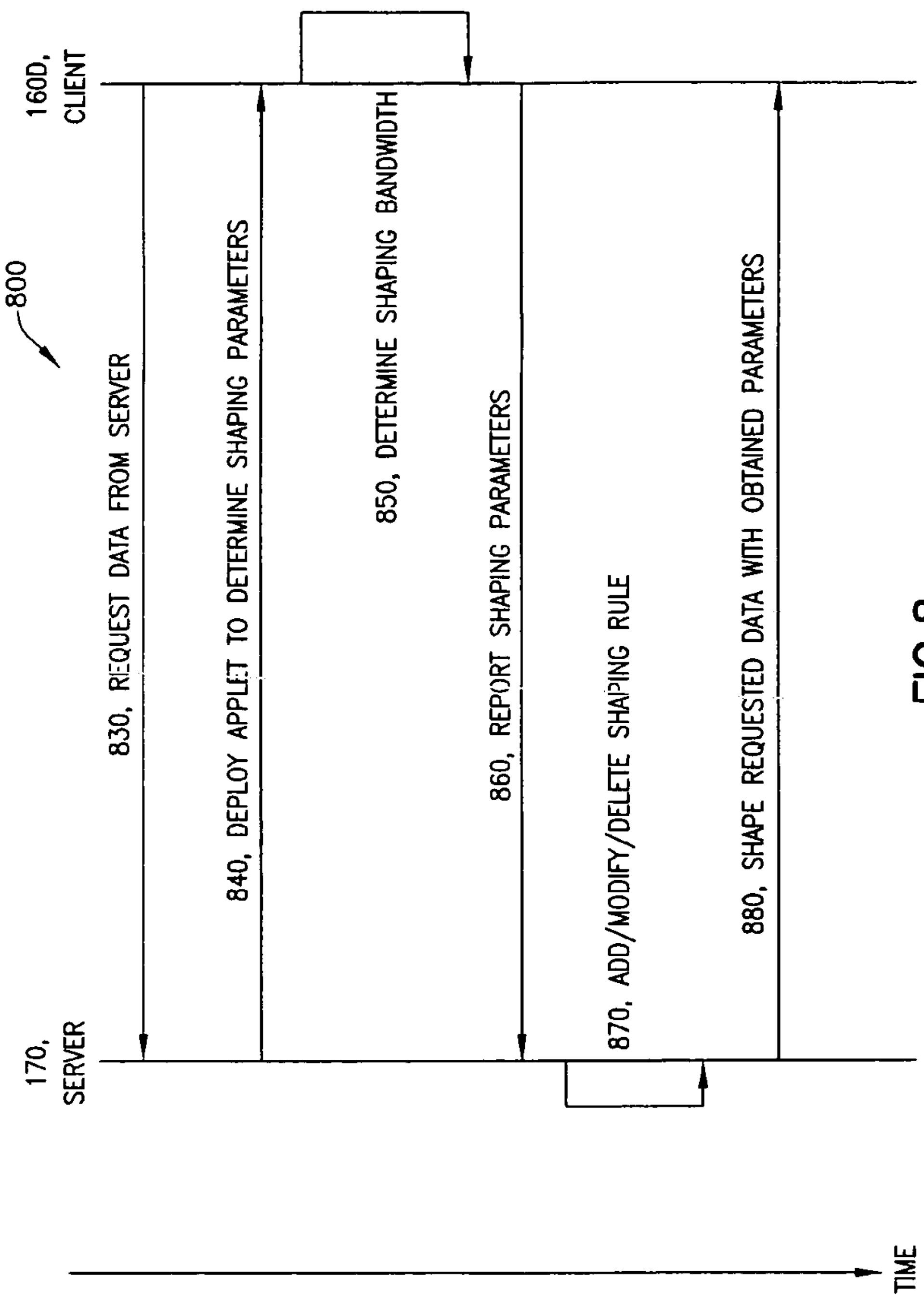

800
160D, CLIENT
170, SERVER
830, REQUEST DATA FROM SERVER
840, DEPLOY APPLET TO DETERMINE SHAPING PARAMETERS
850, DETERMINE SHAPING BANDWIDTH
860, REPORT SHAPING PARAMETERS
870, ADD/MODIFY/DELETE SHAPING RULE
880, SHAPE REQUESTED DATA WITH OBTAINED PARAMETERS
TIME

FIG.8

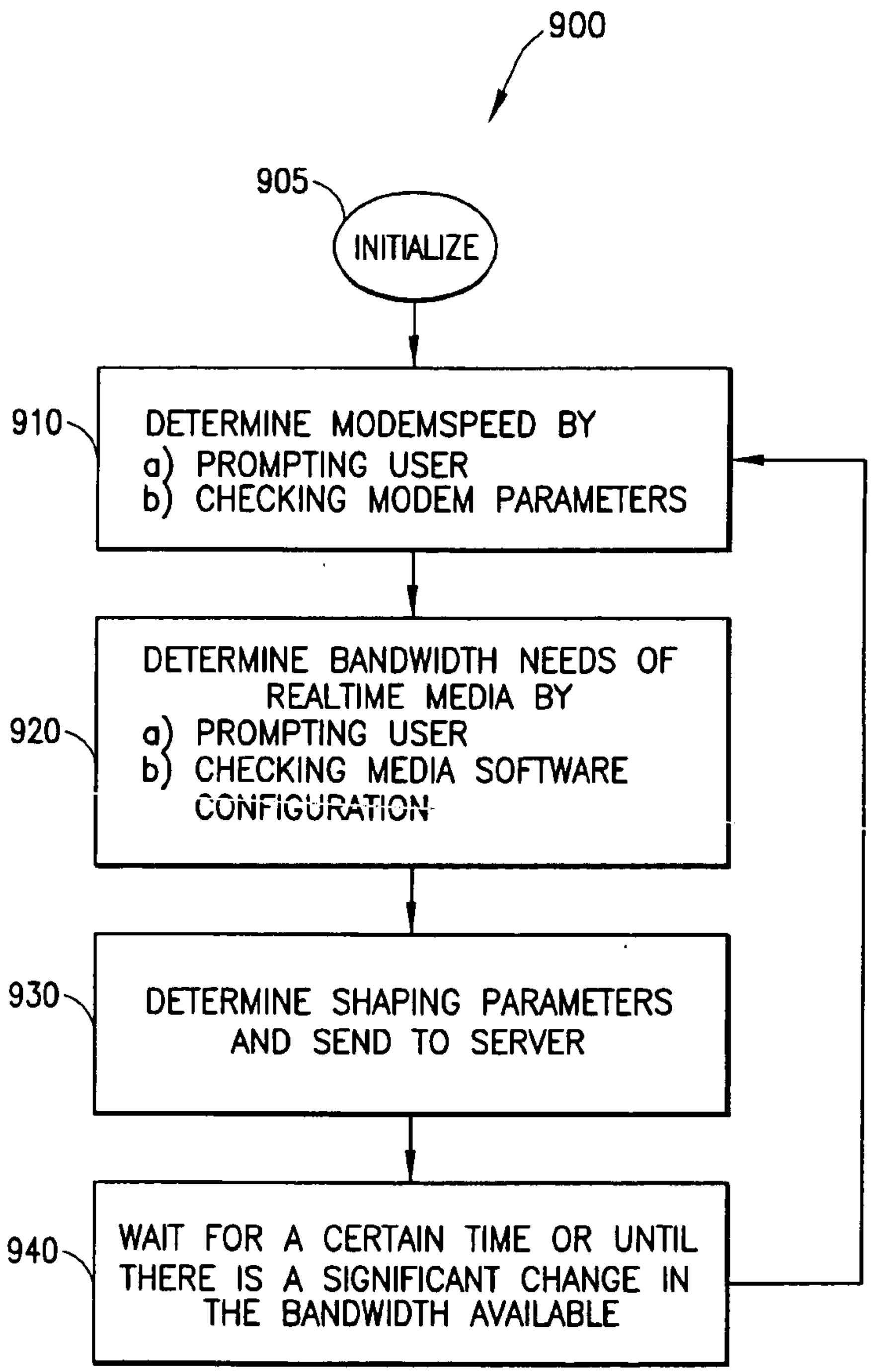

900
905
INITIALIZE
910
DETERMINE MODEMSPEED BY
a) PROMPTING USER
b) CHECKING MODEM PARAMETERS
920
DETERMINE BANDWIDTH NEEDS OF REALTIME MEDIA BY
a) PROMPTING USER
b) CHECKING MEDIA SOFTWARE CONFIGURATION
930
DETERMINE SHAPING PARAMETERS AND SEND TO SERVER
940
WAIT FOR A CERTAIN TIME OR UNTIL THERE IS A SIGNIFICANT CHANGE IN THE BANDWIDTH AVAILABLE

FIG.9

EFFICIENT CLASSIFICATION, MANIPULATION, AND CONTROL OF NETWORK TRANSMISSIONS BY ASSOCIATING NETWORK FLOWS WITH RULE BASED FUNCTIONS

FIELD OF THE INVENTION

This invention relates to the field of identifying and controlling packets sent to and received from a networking environment, particularly one or more of the following: the internet, intranet, cable, and any other of packet switching networks. More specifically, the invention relates to a way to control how packets are transmitted from an application to the network and how packets received from the network are passed to the application.

BACKGROUND OF THE INVENTION

Applications using the Internet for transmission of data and media have huge business opportunities and controlling how information is sent from an application to a network and passed from a network to an application is a critical element. For electronic business it is important that data is manipulated before it is sent to an untrusted network and manipulated after it has left the untrusted network such that one or more of privacy, authenticity and data integrity is assured. For real time information like audio and/or video, it is more important to be able to guarantee an acceptable level of service to make it a successful business. For pervasive computing applications it is critically important that a new class of user machines, such as thin clients and application-specific Tier 0 devices, with widely varying resource capabilities are able to avail of Internet application services without excessive demands on their limited resources.

"Internet Media" transmission includes sending media packets (containing any of the following: n-dimensional images, animation, music, text, movies, video shots, still pictures, voice, data, etc.) over packet switching networks (e.g., a wide area network—WAN—and/or local area network—LAYN) between two or more computers with special application software. Internet Telephony is a particular version of Internet Media where packets contain voice information (and sometimes video information). When the voice processed by an input device is captured at a source computer, an application running on the source computer will transform the continuous voice analog signals into a series of discrete digitally compressed packets. There are some well known industry standards to define this transformation process and the format of these discrete (often digitally compressed) packets, for example, PCM, GSM, G.723, etc.

There are other known processes defined by standards (e.g., IP, UDP, TCP and RTP protocols) to augment the packets with necessary headers and trailers so that these packets can travel over the common packet switching network(s) to a destination computer. With these headers and trailers, packets usually travel over the packet switching network(s) independently. At the destination computer, arriving packets are stored in a buffer and are then transformed back into the form which is close to the original analog signal. The same industry standard (e.g., PCM, GSM, G.723, etc.) defines this transformation.

Enhancing a network transmission over a non trusted network with security features comprises but is not limited to any one or more of the following:

- message integrity allows a recipient of a transmission to verify that the contents of the transmission have not been altered by a third party. It usually involves the computation of a Message Authentication Code (MAC) that is computed over the content of the transmission.
- privacy guarantees that no unauthorized party can get access to the information. It involves encryption at the sending end and decryption at the receiving end.
- authentication allows a recipient of a transmission to verify the ID of the sender.

STATEMENT OF PROBLEMS WITH THE PRIOR ART

Quality is a serious problem in sending media over packet switching networks, including Internet and Intranets. This problem comes from the two general characteristics of packet switching networks, namely: (A) most users are connected to the Internet over a low bandwidth link (e.g. dialup over a phone line to the Internet Service Provider); (B) a large number of users may connect to the Internet using heterogeneous resource-limited machines and devices, e.g., thin clients, handheld devices, set-top boxes, and Web appliances; (C) currently there is no standard that is generally implemented and allows to differentiate priorities of real time traffic from non real time traffic.

Generally, the prior art systems do not control well how packets are transmitted from an application to the network and/or how packets are received from the network and passed to the application. Here control includes but is not limited to the following: controlling the temporal spacing and the temporal frequency of packets, controlling the security features (encryption, message integrity, authentication) of one or more packets. This lack of control causes several problems, among them packet transmission delay.

For two-way Internet media transmission, long delays are fatal and packet losses also have an impact on the quality of the transmission. Delays occur when packets are buffered, which happens usually in routers, where packets from different incoming links arrive at the same time and have to be multiplexed on fewer or slower outgoing links.

One prior art system for reducing delays is describe in RFC 2205 "Resource ReSerVation Protocol". It defines a protocol to establish a reservation for specific transmission sessions on a given path. This enables routers to give packets belonging to a reserved flow a higher priority. The consequence is that they can be transmitted from one router to the next with little or no queuing. This reduces the delay for such packets significantly. The problem with this prior art system is that it doesn't scale very well, since the router needs to store the priority for all of these sessions. In addition, there is a current lack of a universally accepted policy that restricts everybody from establishing a reservation for a session.

Another prior art system is described in the IETF draft "Differentiated Services". It defines a more scaleable way to give different priorities to different flows. However, this technique is not yet mature enough to be standardized, let alone to be implemented.

Both of the prior art systems are implemented on network equipment (routers) within the network. Since the Internet is not one homogeneous, centrally administered network but comprises many different networks that are under the administrative control of different organizations, it is currently not possible for an end system to obtain a better than best effort quality over the Internet.

An example of a typical prior art networking system 100 for transmitting real time information, including voice and data, and non real time information, is shown as a block diagram in FIG. 1. The networking system 100 comprises a plurality of computers (generally 160) that are connected to one or more networks 130 through well known network connectors such as modems and/or LAN adapters 150. The computers 160 typically can be any generally known computer system, such as a personal computer (like an IBM ThinkPad) or workstation (like an IBM RS6000), or a device with possibly limited memory and a possibly less powerful central processing unit like a set-top box, a hand held device such as a Palm Pilot, or other Web-based application devices. For a one way communication, one computer 160 would be the source computer 160S originating the transmission of information and one or more of the computers 160 would be the destination computer 160D that would receive the information. However, in many applications, both the source computer 160S and the destination computer 160D functions are contained in a single computer, e.g. 160, that can perform both transmission, sending and receiving functions, to enable point to point two way, one to many, and/or many to many communications. The computers 160 will have well known input and output devices like microphones 131, speakers 132, keyboards, mice, cameras, video recorders, screens, recorders, music instruments, pen inputs, touch screens (not shown), etc. The combination of one or more multimedia interfaces 133, e.g. a sound card and/or video card 133, network interface software 134, and one or more network connections 150 converts signals from an analog continuous form 135 to a digital (and typically compressed) packetized form 120. Through the network connector 150, the packets are exchanged over the networks 130 between the computers 160.

The computers 160 might also use a network client 195 (e.g. a World Wide Web browser, an ftp client and other well known clients) to interact with a network server 170 (e.g. a World Wide Web server, an ftp server and other well known servers). The computers 160 therefore behave as a client to the data server 170 and as source and/or destination in the media transmission between them (the server 170 and client 160).

The network(s) 130 can be any type of packet switching network which include but is (are) not limited to the Internet, intranets, extranets, wide area networks (WANs), local area networks (LANs), phone networks, and/or any combination or interconnection of such networks. Typically these networks comprise access points 140, routers 110, and network links (typically 175). Network links 175 connect these routers 110 and access points 140 to form the network 130. These routers 110, access points 140, and network links 175 are typically operated by one or more Internet service providers (ISP). Access points 140 are the gateways to outside world of the closed network. Various computers 160 can access the network 130 via access points 140 by well known connections including: dial-up connections, dedicated line connections, cable connections, satellite connections, and other forms of well known connections. The computers could also be attached to the network with a wireless interface based on transmission of radio waves, infrared and other well known interfaces.

Known standard protocols (IP protocol, PPP protocol, LAN protocol, etc.) support various computers 160 to exchange data and messages independently of the connection being used between the network connectors 150 and the access points 140. Particularly, User Diagram Protocol (UDP) and Real-Time Protocol (RTP) provide the ways for computers to exchange real-time Internet media packets over the network 130. Other known standard protocols (TCP protocol, HTTP protocol) are better suited for transmission of non real time information like data transmission.

In the example of FIG. 1, the destination computer 160D is receiving real time UDP/RTP packets 120 over the network 130. At the same time the destination computer 160 downloads data from the World Wide Web server 170. Since both flows of packets have the same destination, they merge at a merging point 190 either on a router 110 within the network or at the access point 140 that connects the Destination Computer 160D to the network 130. Merging the flows means that packets 120 and 180 are interleaved temporally. As it is usually not possible to give a higher priority to the real time packets, every packet will be put at the end of the merging buffer. This can add a significant delay to the packet transmission time and reduce the quality of the real time transmission to an intolerable level. Since the network is not under the control of the administrators/operators of the computers 160 and 170, the quality of the media transmission 120 can not be controlled is usually not acceptable.

Security is a addressed in a prior art system called the Secure Socket Layer protocol (SSL). It is also known under the name Transport Layer Security (TLS). It allows the provision of security features to an individual TCP connection. However, it uses a specific API (Application Programmer Interface) that is different from the interface usually used to transmit and receive data to/from the network. This means that existing applications that want to make use of security features need to be modified to use this specific API. Since SSL is based on a reliable transport mechanism it is only specified for TCP transmissions.

Another standard dealing with security issues is called IP Security (IPsec). It defines how to setup a security association or a secure tunnel between two points connected to a network. Packets are classified for transmission over the tunnel based on source port, destination port, source address, destination address, and the protocol. However, it is not possible to make the classification dependent on a process/thread group or ID or a specific sending or receiving application. Associating a packet with the required security features is usually done using a filter that finds the required security features based on a lookup of the classification parameters of the packet. This can be a computationally expensive process.

OBJECTS OF THE INVENTION

An object of this invention is a general way to efficiently control and manipulate packets sent to the network and pass packets received from the network to the application based upon one or more criteria.

An object of this invention is a general way to efficiently control and manipulate packets sent to the network and pass packets received from the network to the application based upon one or more of the following criteria: packet source address, packet destination address, packet source port, packet destination port, protocol type of the packet, and the type of application that sends or receives the packet.

An object of this invention is a system and method to provide better quality to an end-to-end Internet media transmission between two points which are connected by one or more packet switching networks, by rate shaping all other transmissions that are destined to one of the two points.

An object of this invention is a system and method to reduce the delay, the delay variation, and the loss rate of an end-to-end Internet media transmission between two points which are connected by one or more packet switching networks and exchange other data as well by rate shaping all other transmission having the same destination.

An object of this invention is a system and method for a source computer, and a destination computer, to negotiate and determine an aggregate rate to which all transmission from the source computer to the destination computer is limited.

An object of this invention is a system and method that reduces the transmission delay variance and the packet loss rate of a transmission over one or more packet switched networks from a source to a destination computer that are attached to these networks by rate shaping the transmission at the source computer.

An object of this invention is a system and method to control the temporal spacing and frequency of packets sent to the network from an application and packets received from a network and sent to the application of a set of one or more network transmission sessions, according to an aggregate policy, so that all transmission sessions are treated alike and fair.

An object of the invention is to control encryption, authentication, and message integrity of packets sent to and received from the network based on an efficient lookup/association of the security features and the packets.

An object of the invention is to give similar security features to a specific set of packets sent to or received from the network.

An object of the invention is to give similar security features to a specific set of packets sent to or received from the network, that can be identified using one or more of the following: packet source address, packet destination address, packet source port, packet destination port, protocol type of the packet, and the type of the application that sends or receives the packet.

An object of the invention is to enhance a software application with features including but not limited to temporal spacing and frequency of packets and security like encryption, authentication and message integrity without having to modify the application.

SUMMARY OF THE INVENTION

This invention is a system and method for classifying, manipulating, and/or controlling communications, e.g., packets transmitted over a network. A computer, e.g. a server, connected to one or more networks contains applications that communicate over the network through a network interface by sending and receiving packets. Each application is connected to the network through one or more sockets to enable this communication. The computer also comprises one or more rule sets of one or more rules. A socket set of one or more of the sockets is associated with only one of the rule sets. The rules in the rule set are used to control one or more of the packets communicated by the applications communicating over the socket(s) associated with the respective rule set. In one preferred embodiment, the rules in the rule set can be added, modified and deleted to classify, manipulate, and/or control the communication of the packets, e.g. to control the rate at which the packets are sent or to provide certain security functions.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a prior art network.

FIG. 2 is a block diagram of a network showing a novel server and client computer connected to the network.

FIG. 3 is a data structure of the present invention showing rule sets associated with socket sets in the server.

FIG. 4 is a block diagram of the functional blocks of one preferred embodiment of the present invention.

FIG. 5 comprises FIGS. 5A, 5B, and 5C, which are flow charts showing "add rule", "modify rule", and "delete rule" processes, respectively.

FIG. 6 is a flowchart showing the process how sockets are classified and associated with rules.

FIG. 7 is a flowchart of the rate shaping process.

FIG. 8 is a diagram of the message flow between the client and server communicating client constraints.

FIG. 9 is an optional client process that is executed on the client to communicate client constraints to the server.

DETAILED DESCRIPTION OF THE INVENTION

FIG. 2 is a block diagram of a networking system **200** showing a network server **170** (e.g., a World Wide Web server) that contains a novel packet control mechanism (packet controller) **202**, a client/destination **160D** which comprises an optional novel client bandwidth determinator **210**, and a client/source **160S** connected to the network **130**. Note that components of the networking system **200** that are in the prior art were described in FIG. **1** and have the same numerical designators.

Client **160D** includes but is not limited to any of the following computing devices: A computer, like a PC (e.g., IBM Aptiva), workstation (e.g., IBM RS/6000) a labtop/notebook (e.g., IBM Thinkpad), a network computer, or a set top box, a low end device like a hand held device (e.g., a PalmPilot, a smart cellular/desktop phone, or other application-specific devices) or any other general purpose computer. The computer is attached to the network **130** over any type of interface like wireless, dialup using a modem, ISDN, ADSL, cable, etc. The client/destination **160D** acts both as a receiver of media like audio, voice and/or video, and as a client **195** to a network server **170**. An example client program would be a World Wide Web browser, the client side of a File Transfer Protocol (FTP) connection, a PointCast client, etc. The network **130** could be any packet switched network such as the public Internet or private corporate intranets.

The client **160D** optionally contains a client bandwidth determinator **210** that allows local determination of the bandwidth that is available for the network server to download data to the client. Bandwidth includes but is not limited to any one or more of the following parameters: peak rate and average rate at which packets are sent from the network server to the client, maximum burst sizes at which packets can be sent at the peak rate, and the maximum length of the packets. These control parameters are described in more detail in FIG. 3. In the following, we use the term "rate" interchangeably with bandwidth.

The network server **170** can be any of the following: A general data server (e.g. ftp server), a World Wide Web server or proxy, a database server, a mainframe, or any other general purpose computer used in a generally well known server application, etc. It could also be a streaming media (audio or video) server (e.g., RealAudio or RealVideo) that streams multimedia data to requesting clients on demand. It is also possible that the network server is used as a front end for a database server or mainframe and allows such a device to be indirectly connected to the network (e.g., the Internet). Note that the server could transmit data to clients using a variety of transport protocols, e.g., Transmission Control Protocol (TCP) or User Datagram Protocol (UDP).

The network server 170 contains a packet controller 202 that can includes but is not limited to on or more of the following mechanisms that: control the timing and the frequency at which packets are sent to the network 130, control the timing at which packets received from the network are passed to the application, control the size of packets sent to the network, control the size at which data is passed to the application, and control the security features of packets sent out to the network and received from the network. The packet controlling mechanism 202 has a data structure 300 and the functional software blocks 400 that work together to allow efficient association of packets with the controlling mechanisms that should be applied.

FIG. 3 is a block diagram representation of a data structure 300 of the present invention showing rule sets associated with socket sets in the network server 170. A network server like a WWW server software usually creates several processes or threads 300 whose instances are depicted as 300*a*, 300*b*, 300*c* and that implement a network protocol to receive and transmit data (e.g., the HTTP protocol for WWW servers). In the following we will use the term application to include both process and thread.

Each of the applications uses a well defined interface to receive data from the network and transmit data to the network. In most popular operating systems, this interface is called a socket 310, whose instances are depicted as 310*a*–310*f*. In the following we call a socket any abstraction or service of an operation system that allows the sending of data to a network and allows the receipt of data from a network. Therefore, the term socket comprises but is not limited to sockets as used in UNIX and derived operating systems like AIX, FreeBSD, NetBSD, Ultrix, IRIX, Linux and other well known operating systems, OS/2, OS390, and winsock or socket (API) as used in Windows 3.x, Windows 95/98 and Windows NT. A socket 310 usually contains network protocol state 315 that is dependent on the protocol and can comprise but is not limited to any or more of the following parameters: amount of data in the socket buffer, socket blocking or non blocking, high water and low water mark for the socket and other well known parameters. The network protocol state instances are depicted as 315*a*–315*f*.

A rule set database 340 has rule sets 330 whose instances are depicted as 330*a*, 330*b*, 330*c*. In a preferred embodiment, the rule sets 330 are connected using a linked list 350. Iteration through the rule sets is therefore possible by walking down the list. In alternative embodiments, in addition to the linked list, a hash table can be used to find a specific rule more efficiently. The hashing is carried out on parameters such as the source port, destination port, destination address, and source address, and the protocol. Every rule set can be associated with zero or more sockets. An example of hashing is to form a long number out of the source/destination address (usually 32 bits), source/destination port (usually 16 bits), and the protocol type (usually 8 bit), yielding a 104 bit number by just appending the bits of all the field. A hash function, e.g. a modulo function reduces this big number to a smaller set of hash buckets (e.g. 1021 buckets). Since the number of rules in the system is relatively small (smaller than the number of hash buckets) there is a good chance that each bucket contains one rule that can be found using the computationally efficient hash function. More details to hashing can be found in "Introduction to Algorithms" by Corment, Leiserson Rivest, published by Mc Graw Hill.

In a preferred embodiment, a rule set (e.g. rule set instances 330*a*, 330*b*, 330*c*) comprises a data structure 350 depicted as 350*a*, 350*b*, 350*c* comprising any one or more of the following: flow parameters (typically 362), control parameters (typically 364), and state parameters (typically 366).

For example, a rule could comprise a source address X and a destination address Y (these could be IP addresses, IPX or any other well known types of addresses) as flow parameters (meaning that all other flow parameters are wildcards). Therefore, the rule would apply to all transmissions from X to Y and Y to X. As an example, it could also contain the following control parameters: key W, encryption algorithm Z, applicable to outgoing transmissions for every other packet. Therefore, every other packet coming from X and going to Y would be encrypted with encryption key W, using the encryption algorithm Z. In this case, the rule state would consist of a single bit that is toggled with the sending of each packet and that indicates if encryption needs to be applied or not.

Rules can also be classified into types. Rule types comprise but are not limited to any one or more of the following: rule types controlling the frequency and spacing with which packets or data is delivered, rule types controlling security features of a transmission. Rule types can apply on the sending side, controlling packets sent to the network, and on the receiving side, controlling how packets/data is delivered to the application.

The flow parameters 362 describe the extent of the rules, or more specifically which packets need to be controlled. They comprise, but are not limited to, any one or more of the following: source port, source address, destination port, destination address, protocol, the type of sending/receiving application. Any one or more of the former parameters can be a wildcard, denoted by a * (asterisk). As an example, the following flow {source port=*; source address=9.2.23.129; destination port=80; destination address=9.2.24.4; protocol =TCP, application=*} would comprise all TCP connections with IP address 9.2.23.129 and any port on one end, and IP address 9.2.23.4 and port 80 on the other end. That is, source and destination port/address are not related to the direction in which the packets travel that are controlled, but are needed to achieve a mapping with the socket which uses these parameters.

Using that many parameters to describe a flow is usually not possible for systems located in the network (not end systems) that try to implement similar control functions (e.g. a router, or Packeteers PacketShaper). The five parameters source/destination address, source/destination port and the protocol type can be found in packets transmitted by protocols like TCP/IP, UDP/IP and other well known protocols. Therefore, these packets can be identified by a system located within the network. However, the parameter application type is usually not contained in these packets and cannot always be inferred from the other parameters in general. Hence, a router or packet shaper within the network could not get this information by just looking at the packets and could therefore not necessarily perform a classification based on application type.

The control parameters 364 and control state 366 are used to control and manipulate the sending and receiving of packets. In a preferred embodiment, the control parameters 364 comprise any one or more of the following: an indication if the rule applies to the sending side, the receiving side or both sides; an indication, which part of the software (like a function, a procedure, a macro, calling a method on an object or any other well known methods to invoke a piece of software code) needs to be executed; and the parameters necessary for the execution of that code. The control state is usually needed to keep variables between two executions of the respective piece of code.

Controlling and manipulating packets includes but is not limited to any one or more of the following rule types:

Controlling the frequency, the interval, and the size of packets sent to the network and of packets received from the network and being sent to the application, providing mechanisms for privacy, message/packet integrity and authentication.

In a preferred embodiment, the frequency/interval rule type controls the frequency and the interval of packet delivery by using a "leaky bucket" scheme. The leaky bucket scheme is specified in ITU-T recommendation I.371, "Traffic Control and Congestion Control in B-ISDN", March 1993. The leaky bucket scheme uses 2 leaky buckets; one to monitor the peak rate and one to monitor the average rate. The leaky bucket scheme is used to determine if a packet can be delivered or if the packet has to be delayed. In the case of a rule applying for sending, delaying means leaving the packet in the socket buffer. In the case of a rule applying for receiving, delay means not delivering the packet to the network or not delivering it to the application. A scheme with two leaky buckets allows to ensure that the transmission never exceeds a peak rate, and that a long term maximum average is not exceeded. The detailed implementation of a preferred embodiment with 2 leaky buckets is shown in FIG. 7. The control parameters in this case would comprise a peak rate p, an average rate a, a maximum peak rate buffer depth max_pdepth, a maximum average buffer depth max_adepth, and a maximum packet size S. These parameters are all depicted and described in more detailed in FIG. 7.

As an example, the control parameters are shown that apply to a rule guaranteeing the following: that the maximum bit rate of a transmission does not exceed 5 kb/s, that the average bit rate is limited to 1 kb/s, that the maximum burst of packets (packets transmitted at the peak rate) does not exceed 1024 bytes, and that the maximum packet size is 512. The control parameters in this case are: p=5 kb/s; a=1 kb/s; S=512 bytes; max_pdepth=512 bytes; max_adepth=1024 bytes, and a pointer to a piece of code that implements the leaky bucket scheme as shown in FIG. 7.

Rules can also be used for controlling security. These rules are called security rule types. The security rule type controls the security of a transmission by applying security features on some or all of the packets. On the sending side such security features include but are not limited to any one or more of the following: encrypting packets with the purpose of guaranteeing privacy to a transmission, adding a message authentication code (MAC) to a message with the purpose of allowing the recipient to verify the integrity of the message, and signing the message with the purpose of allowing a recipient to be able to verify the authenticity of the sender. On the receiving side such security features include but are not limited to any one or more of the following: decrypting a message, or not passing unsecure, non-encrypted messages on to the application, verifying the message integrity and discarding a message/packet if it has been altered, verifying the authenticity of the sender and discarding packets that come unauthenticated.

As an example, control parameters are shown that are used on the receiving side to decrypt and authenticate messages and discard messages that have been altered: Decryption algorithm=DES; hash function=MD5; authentication algorithm=RSA, and a pointer to a software function that can be described with the following pseudocode (the control state would include the actual keys needed like RSA key and DES key):

```
processPacket(packetPayload) {
    verify the RSA signature of the payload using my RSA key;
    if signature does not match then discard the packet;
    create a MD5 hash;
    compare result with the hash contained in the packetPayload;
    if it does not match then discard the packet;
    decrypt the packet using the DES key;
    notify application to pick up the decrypted payload;
}
```

The control state parameters are initialized in step **710** when a new rule is added to the rule database. In the case of frequency/spacing rules, the state is usually initialized to fixed values as shown in FIG. 7, step **715**. For security rules, the control state can comprise keys, that can be established using well known key exchange protocols like SSL, ISAKMP/Oakley and other well known key exchange protocols.

This invention forms a bidirectional association between rule sets and sockets since a socket is always involved in sending and receiving packets to and from the network. The bidirectional association **330** is depicted as **320*a*-320*f***. In the example of FIG. 3 the rule set **330*a*** is associated with socket **310*a***, **310*b***, **310*c*** using the associations **320*a***, **320*b***, **320*c*** the rule set **330*b*** is associated with sockets **310*d***, **310*e*** using associations **320*d***, **320*e*** and rule set **330*c*** is associated with socket **310*f*** using the association **320*f***. The details of how to create, use, and delete these associations are described in FIGS. **4**, **5**, and **6**. Having a bidirectional association between sockets and rule sets is important for the following two reasons:

1. Efficiently finding the sockets associated with a rule set is important when a rule is deleted (as described in step **442** and following of FIG. 4), because all the associations between the rule set and possibly many sockets has to be removed. Instead of having to iterate through a list of sockets to from which socket is associated with the rule set, it is preferable to be able to find the sockets quickly, starting from the rule. This can be efficiently done using the one to many association **320**. In a preferred embodiment, this association can be implemented as a linked list to sockets that is stored in the rule set. Each socket only needs to have a single pointer to the rule set.
2. Efficiently finding the rule set associated with each socket is important to find the control state **366** and control parameters **364** when the socket needs to send out a packet., or when a packet is received, since these parameters are needed to manipulate and control the packets (as shown in FIG. 7 for the example of spacing/frequency type of rule). Finding the rule set associated with a socket is also done when a socket disconnects as shown in step **444** and following of FIG. 4.

FIG. 4 is a block diagram of the functional blocks **400** of one preferred embodiment of the present invention. In one preferred embodiment these blocks are implemented in software. It shows the functional blocks necessary to create, modify and delete the data structure described in FIG. 3.

Each functional block (represented by a square) contains a clearly defined module. This module is activated by a trigger event represented by a circle. The circles **440-449** represent typical trigger events. Trigger events include but are not limited to any one or more of the following: A trigger event can be caused by an application that manipulates a rule

(440–442), an application that changes the state of a socket (443, 444), an event that causes an attempt to send data to the network (445–447), or an event that is caused by the receipt of a packet or that causes an attempt to receive data on a socket (448–449). In a preferred embodiment each trigger event is a call to a function (or more general a piece of software) and the parameters of the trigger event are the parameters that are passed in the function.

The functional block 450 validates if the manipulation of a rule (caused by a trigger event 440–442) is valid. This block is very specific to the environment and the security constraints the invention is placed in. In the following we describe possible policies for rule validation for the trigger events 440–442.

An add rule (440), modify rule (441), delete rule (442) trigger event can be issued by an application (e.g., a WWW server that has determined the available bandwidth to a client), or a management policy that e.g. defines security features for certain flows. In the former case, a network server gets a request for the transmission of data and, based on previous knowledge or by learning how much bandwidth is available on the link to the requester, triggers the adding of a rule 440 that limits the bandwidth used by the transmission. This bandwidth can change during the transmission and the rule can be updated by triggering a modify rule event 441. When the transmission is over, the rule is deleted (trigger event delete rule 442).

In the latter case, e.g. a management policy can be implemented by a user or a software agent, that sets up the security rules or they can be stored in a file together with temporal information on when rules need to be added, changed, deleted. As an example, a policy can restrict clear text transmission of telnet sessions to a server. This is achieved by adding a rule that contains the

TCP protocol, three wildcards for the foreign port and the P addresses, and the local TCP port used for telnet. All socket connections going to that port would then mandate packet encryption. Unencrypted packets on the receiving side would for example be discarded by the rule.

The rule validation 450 comprises but is not limited to verifying if a rule is "grammatically correct" (e.g. that there are no nonexistent protocols, negative port numbers, IP addresses of incorrect length, etc.), if the issuer of the trigger has the permission to do so etc. In one security context, manipulating rules might be restricted to a Webmaster, in another context the WWW server application might change the rules itself. Some users/applications might not be allowed to setup rules other might only be allowed to change rules. If the validation is not successful an error is returned to the calling entity, otherwise the trigger event is passed on the rule processing block 500. As an example, the following pseudo code validates the trigger events 440–442.

```
validate(triggerEvent) {
  IF issuer of triggerEvent is authorized AND
  IF time to apply triggerEvent is ok AND
  IF trigger event is 'grammatically' correct AND
  . . .
  THEN pass on trigger event to block 500
  ELSE ignore trigger event and return
}
```

The algorithm for the functional block 500 is described in more detail in FIG. 5 and is used to add, modify and delete rules shown in FIG. 3 and the associations they have with sockets.

The trigger event 'connect socket' 443 occurs when an application on the server establishes a TCP connection or when it creates a connected UDP socket. This newly connected socket has then to be matched with every rule to check for a possible necessary association. This is done in the functional block 600 which is described in detail in FIG. 6.

Upon the trigger event 'disconnect socket' 444, which occurs e.g. when a TCP connection terminates or a UDP socket is disconnected, the socket is disassociated from any rule set. Disassociation here means breaking up the bidirectional association 320*a*–320*f* shown in FIG.

3. The following pseudo code serves as an example.

```
disconnectSocket(s) {
  IF socket is associated with a rule set r
    THEN
      find the pointer to socket s in the linked list;
      remove the pointer from the linked list;
      remove the pointer from socket s to rule r
}
```

In a preferred embodiment, a timer is used if there are shaping/frequency rules in the system. This timer times out periodically and triggers a 'timeout triggered send' 445. This can be useful e.g. to send out packets that have been delayed before as shown in FIG. 7 step 720 when the leaky bucket test is not true. For other types of rules, a similar timeout mechanism might be used as well.

Attempting to send a packet includes but is not limited to the following: a timeout triggered send 445 occurs when the timer associated with the control mechanism and provided by the operating system fires (i.e., expires). It is enough to have a single timer, that fires and resets itself automatically, in the system. The granularity of the timer determines the maximum frequency with which controlling occurs, in the case where it is the only triggering event. On the other hand, a finer granularity of the timer means that more CPU time is spent iterating over the sockets and applying the control mechanism.

The application triggered send 446 is issued by an application like a WWW server (after a browser has requested to receive a certain WWW page) or data part of an FTP server (after an ftp client has requested a certain file) that tries to send data to the network.

A protocol triggered send is caused by specific protocols. For example, the TCP protocol updates its internal state, whenever it receives an acknowledgment from the peer. An acknowledgment signifies that the peer has successfully received data and that the sender/server might be able to send more. The TCP protocol then evaluates if sending more data is permitted and issues a protocol triggered send.

All three send trigger events (445–447) have to pass through the control mechanism 700, which applies the control function contained in the rule. A specific example, where the control mechanism limits the rate on the sending side, is provided in more detail in FIG. 7.

The receive trigger event (448) occurs when the application wants to receive a message on a socket, while the packet arrival trigger event (449) occurs when a packet arrives from the network for a socket. Both these trigger events have to pass through the control mechanism 460, which decides how the arrived packets are processed with respect to the associated rules before being received by the application. For example, shaping rules for incoming packets may specify that the server accepts no more than a certain rate of FTP or HTTP connection requests (i.e., TCP connection setup packets). Such rules can reduce the possibility of denial-of-service attacks, for example, and also provision a portion of server receive resources under policy control. The associated rules may also specify the rate at which data contained in arriving packets is delivered to the application socket for subsequent receives.

FIG. 5 comprises FIGS. 5A, 5B, and 5C, which are flow charts showing how the rule processing functional block handles "add rule", "modify rule", and "delete rule" trigger events. These trigger events and why and when they occurred are described in FIG. 4.

FIG. 5A describes the processing of an add rule event. An add rule event 440 carries control and flow parameters to create a new rule as described in 350 of FIG. 3. Step 511 searches for a rule with identical flow parameters. If the search was successful (branch true of 512) then an error is returned 513, as duplicates are not allowed (instead, if a rule for a flow needs to be changed the modify rule trigger 441 has to be used). If the rule has a different set of flow parameters from all other rules (branch false of 512), it is added to the database in 514. Note that it is possible that two rules with different flow parameters match for the same socket (because of the use of wildcards). In this case, the better matching rule takes precedence and is associated with the socket. This is described in the following.

The following steps show for each socket if it needs to be disassociated with any previous rule and associated with the new rule or be associated with the new rule for the first time. Disassociating a socket from a previous rule to associate it with a new rule is necessary if the new rule has flow parameters that match better with the socket parameters than the flow parameters of the old rule. To test for a better match a cost function is defined that determines how costly a match is. Every rule in the system should always be associated with the socket that yields the lowest cost match. Finding the lowest cost match is described in detail in FIG. 6 where the reverse initial situation exists: a socket needs to be matched with the best matching rule instead of a rule needing to be matched with all sockets that match.

Step 515 shows that an iteration over all sockets is performed. Most operating systems keep a linked list containing all sockets. Iterating over the sockets means just following the elements of the linked list. Alternately, a hash table may also be constructed to efficiently identify the set of sockets that most closely match the flow parameters in the new rule. In the following the current socket is denominated as socket s.

Step 516 checks if the new rule r needs to be associated with the current socket s under consideration. Two functions are defined (match and cost) which are used in the test. The function match(s,r) is true if for a socket s if the IP source address, IP destination address, source port, destination port, protocol, and application type are identical to the corresponding fields in the flow parameters of the rule with one exception. An * (asterisk) in a field in the flow parameters is always considered to be a match as shown in the following pseudo-code

```
match(s,r) {
IF (s.source_address =r.source_address OR r.source_
    address=*) AND
(s.destination_address =r.destination_address OR
    r.destination_address=*) AND
if (s.source port=r.source_port OR r.source_port=*)
    AND
(s.destination_port =r.destination_port OR
    r.destination_port=*) AND
s.protocol=r.protocol OR r.protocol=*)
THEN return TRUE
ELSE return FALSE
}
```

There are two cases which are treated differently for a match. If the socket was not associated with a rule before

and the flow parameters match, step 517 is performed. If the socket was associated with a rule before, then step 517 is only performed if the new match is better. In other words the new match is better if the cost function for the socket and the rule is lower than the cost function for the socket and the rule it used to be associated with. Step 517 breaks up any old association (if any) by associating the current socket s with the new rule r. The cost function is shown in steps 615–645 and the value it produces is contained in the variable localCost at step 645.

Step 518 checks if the iteration has covered all sockets in which case it is stopped 519 or if there are more sockets left in which case step 515 will find the next socket to be evaluated.

FIG. 5B describes the processing of a modify rule trigger event. The modify rule trigger event is described in FIG. 4. The modify rule event 441 identifies the rule and the new flow and/or control parameters. Step 531 locates the rule using a hash function over the IP source and destination addresses, the source and destination ports, the protocol, and the application type. If the rule is not found (step 532) an error is returned in 533. If the rule is found, the control parameters 364 contained in the data structure 350 of the found rule are updated in 534 using the parameters of the trigger event and a return occurs at 535. After updating of the control parameters, the new control state values have to be verified for correctness. An example where control state parameters need to be updated is shown in FIG. 7.

FIG. 5C describes the processing of a delete rule trigger event 442. As parameters it contains the flow parameters that uniquely identify the rule that has to be deleted. In step 551, the rule is located in the rule database using the hash function over the IP source and destination address, and the source and destination port for this protocol. If the rule is not found in test 552 then an error is returned in 553. If the rule has been successfully located, it is disassociated from all sockets and removed from the database at 554. For all the sockets that have been associated with the rule, at 555 the classifier described in FIG. 6 is used to see if there are any other rules that now match best with the disassociated sockets and a return occurs at 556.

FIG. 6 is a flowchart showing a rule classification and association process 600 that is used to create bidirectional associations 320 between rule sets 330 and sockets 310. It is triggered by a request 443 to classify a newly connected socket and associate it with the best matching rule. The request comprises the following flow parameters {source IP address, destination IP address, source port, destination port, protocol, application} depending on how the system is setup.

In step 605 local variables globalCost and matchingRule are set to zero.

In this preferred embodiment we find the best matching rule by iterating over all the rules and picking the rule with the least cost matching function as shown in steps 620–640. In another embodiment, the hash function might be applied first for the socket parameters as they are, next to the socket parameters where one parameter is replaced with an asterisk (do this for every parameter), next where 2 socket parameters are replaced by asterisks (do this for all possible combinations) then 3,4 and 5 parameters replaced with an asterisk. Depending on how many rules generally are in the database this might be a slower or faster approach to find the best matching socket. Either of the approaches are done in step 610.

The following steps 615–640 determine if the selected rule matches the socket and what the cost associated with the match is. The cost function can yield three types of values: The value zero for a perfect match, meaning that none of the parameters in the rule was a wildcard (asterisk). It can yield a no match as shown in step **630** (no flow). Lastly, it can yield a positive, non zero value that signifies a costlier match the higher the number is. The cost function allows to parameterize the cost of a wildcard match by associating a specific cost to each of the parameters with which the connect function is triggered.

In step **615** the local variable wildcard is set to zero. Step **620** starts with the choosing, of the first of the 5 parameters shown in the box. In subsequent rounds all the parameters are picked one after another. The parameters can be chosen in any order.

Step **625** tests if the corresponding flow parameters of the rule is a * (asterisk). Only if the rule contains a wildcard, the cost function has to be used to determine the overall cost of an eventual match. If the test of **625** is false then the fields of the rule and the socket are compared for equality in step **630**. If the two fields are different, the rule does not match the socket and any fur matching for this rule is not necessary. In this case, the next rule is considered for matching in step **610**. If the two fields are equal step **640** is executed next. If in test **625** the rule field contains a * (asterisk) the cost for such a match is added to localCost of matching this rule with the socket. The cost for a wildcard match can be different for each field of a rule and allows to give priorities in case there are two rules which have an equal number of wildcard matches but where the association with one rule is preferred over the association with the other. The cost of a wildcard match for each field can be statically stored and is obtained in step **670**. Step **640** checks if there are more fields that need to be compared. If so, then the next field is selected in step **620**. If there are no fields left, the socket and the rule have been tested for a matching in all fields and the testing was positive. In the following step **645** the cost of this latest match is compared with the cost of the best match so far. If the cost of this (local) match is smaller than the cost of the best match (global) so far, then this match replaces the former best (global) match in step **650**. Otherwise the next rule is examined for matching. In step **650** the new best match is stored in matchingRule, together with the cost associated with that match (globalCost). Note that the cost of any match that did not involve a wildcard is zero. Since there is only exactly one such match possible (rules with identical flow parameters are not permitted), there is a unique best match possible. Test **655** checks for more rules that have not been examined yet. If there are more rules, the next rule is examined in step **610**. If there are no more rules, the socket can be associated with the best matching rule that is contained in the variable matchingRule **660**. Note that it is possible that no rule matches a socket and that in this case the socket is a prior art socket that is controlled by any rule.

FIG. **7** is an example of a rule control mechanism **700** that implements rate control at the sending side. It could also be applied to control the rate at which packets are passed to the application on the receiving side. It allows to reduce the rate between the network server and the client. This can be useful if the client has other concurrent sessions, (e.g. audio/video sessions) that might be impacted by the server-client transmission if it is not rate controlled.

The packet shaping process is based on a 'leaky bucket' scheme, that uses two leaky buckets. One leaky bucket is used to monitor conformance to the peak rate p, the other one is used to monitor conformance to the average rate a. In the following an intuitive explanation of one leaky bucket is given. The leaky bucket has a certain capacity to store

credits. These credits are measured in bytes. Whenever the shaping timer expires, both leaky buckets are filled with a different amount of credits. The amount of credits added to the peak rate monitoring bucket is T*p, the amount added to the average rate monitoring bucket is T*a where T is the interval of the shaping timer. This is shown in step **710** and explained in more detail below. The bucket cannot be filled more than its maximum capacity (max_pdepth and max_adepth for peak and average rate buckets, respectively). At any time, it is only possible to send as much data as is accounted for in both buckets (cannot send more than the credit available). If data is sent, the credit of both buckets (pdepth, adepth) is decrement by the amount of data sent.

As shown in FIG. **4** the packet shaping process can be triggered by 3 different events. The most common trigger event is an application trying to send data to the network using a connected socket. This is the trigger event **446**. Another trigger event that involves the same order of processing, is a protocol triggered event **447**. Such an event could be a timeout of the protocol used to transmit data (e.g. in the case of TCP, a retransmission timeout causes the TCP protocol to retransmit data), or feedback from the network or the peer that it is ready to receive more data (e.g. in the case of TCP an Acknowledgment from the receiving end, that opens the sending window and can cause new packets to be sent). The third trigger event is caused by the control mechanism itself. It occurs when the control timer expires (step **445**) and causes a filling of the leaky buckets in the case where the rule implements a control mechanism for the rate. Whenever the shaping timer expires the following steps will be performed for all rules. This is achieved by iterating over the linked list containing the rule database shown in FIG. **3**. Step **750** replenishes the content of both buckets. The peak leaky bucket is incremented by T*p, the average leaky bucket by T*a. While in the preferred embodiment, the value of "T" is the same for both the peak leaky bucket and the average leaky bucket, each bucket, could have different values of "T". Since

T is associated with the timer, this would mean that more than one timer has to be employed. Steps **750**–**770** insure that the new content of the bucket does not exceed the maximum bucket size. Step **755** tests if the new content (pdepth) is larger than the maximum depth of the bucket (max_pdepth). If the test is true, pdepth is reduced to the maximum size allowable in step **760**. Steps **760** and **765** check for an overflow of the leaky bucket monitoring the average rate and reduce the depth (adpeth) to the maximum allowable for the average leaky bucket. Then the processing merges with that of **446** and **447** in step **720**. Reducing pdepth and adepth to their maximum values does not impact the packets since these parameters only indicate the amount of credits contained in the respective leaky bucket. Therefore, no packets are discarded in the process.

The first step **720** in the actual shaping process is to verify if new data can be sent to the network without violating the shaping rule. The test is true only if both leaky buckets contain credits (i.e. pdepth>0 and adepth>0) AND if the socket buffer contains data to send (b>0). If any of these conditions is not met the shaping mechanism returns (**725**). The socket buffer does not always necessarily contain data when the shaping mechanism is invoked. For example, after a timeout triggered send the shaping mechanism is invoked but the application might not have any data to send.

Step **730** determines the maximum size of the packet that can be sent out. The maximum size is the minimum of the following parameters: The amount of credits (bytes) available in any of the two leaky buckets (pdepth, adepth), the amount of data in the socket buffer that is ready to be sent out (b), the maximum size of the packet as specified in the shaping parameters (S), and the maximum transmission unit as defined in the network protocol (MTU). The minimum of these parameters defines the length of the packet that can be sent out.

Step 735 then tests if the network protocol allows to send a packet of that size. For example, if the network protocol that is used is UDP (a datagram protocol) then the protocol does not allow to fragment a datagram into two UDP packets. In this case, the shaping mechanism returns (725) and has to wait until enough credits have been accumulated in the leaky buckets that the whole record can be sent out in one shot. Note, that a misconfiguration of the packet size S could prevent any large UDP packets from being sent. Therefore, setting up a shaping rule needs careful consideration with respect to the network protocol it applies to.

Step 740 consists of the actual sending out of a packet of length "len" to the network, involving all lower network protocol layers. In addition, the packet headers also have to be formed according to the network protocol used. In step 745 the leaky bucket state is updated to account for the packet that has been successfully sent. The number of credits in the average and in the peak leaky bucket is reduced by len. Steps 720–745 are then repeated until the condition 720 is no longer true in which case the shaping process terminates the current round.

For the flow diagram comprising the timeout triggered send, a plurality of sockets might be ready to send data at step 720. If the notion of fairness among sockets is not important the set of sockets associated with one shaping rule might be iterated always in the same order. In general, this allows the sockets that appear early in the iteration, to send more data than sockets appearing later in the iteration if the rule implements a temporal constraint (intuitively, the sockets early in the list use up all the credits). This is not a problem for WWW applications where it does not matter if one part or the other within a Web site is built up faster. It is more important that the aggregate rate is not exceeded.

In an embodiment, where fairness is important (i.e. all sockets are supposed to get an equal share of the total shaping rate), a better strategy is to iterate over the socket set associated with one rule in the same order, starting with a different socket in each round. This can be easily implemented by marking the socket that is supposed to send first in the next round. If the socket has data to send in the next round the marker is advanced to the next socket, otherwise the marker remains.

FIG. 8 is a diagram of the message flow 800 between the client and server communicating client constraints. Client constraints can be divided into two categories: constraints of the network bandwidth between the client and the peer it is communicating with, and constraints within the client itself. The latter constraints comprise any one or more of the following: limited memory and limited CPU power as they are common in thin clients, set-top boxes, and hand held devices like cellular phones, pagers, palmtops, etc. For a server to setup a new rule that controls the rate of a connection, it has to have some information what reasonable shaping parameters should be.

In one preferred embodiment, the network server would try to measure the bandwidth that is available between the server and the client. This can be done actively by sending probes into the network, or passively by monitoring the packets that are sent to the client anyway. By keeping track of past available bandwidth, the current bandwidth can be estimated.

In another embodiment, the network server 170 and the client source 160S are collocated. The client source knows the parameters of the transmission that is in process between itself and the client destination 160D. It might also have some prior information about the bandwidth with which the client is connected to the Internet. It then passes this information on to the network server which sets up a new shaping rule to limit the transmission rate between itself and the client/destination to guarantee a good quality of the transmission between the two clients.

In another embodiment, the client/destination 160D and the server 170 communicate to enable the server to setup reasonable shaping parameters. This communication 800 is shown in FIG. 8. It contains a server 170 that communicates with the client/destination 160D to determine the parameters for the shaping rule. The communication usually starts with a request 830 from the client. This can be an HTTP request like GET, or POST, or a request for data within an FTP session. When the server realizes that it is involved in a session with this client it deploys an applet 840 to determine the client constraints. A detailed description of the applet process is given in FIG. 9. This step could be omitted if the client side already has the necessary software (e.g. cached or stored from a previous session, or it could be contained as a plug-in within the WWW browser, etc.).

In another embodiment the server side would just offer a form-like web page where the user can annotate the maximum bandwidth with which he wishes to communicate with the server.

In another embodiment, the server memorizes the bandwidth available for a transmission. When the transmission stops and a new transmission starts to the same client, this information can be used to set up a new shaping rule. Another possibility is to keep track of the available bandwidth to clients that are 'close' to a specific client. Closeness could be measured by the number of bits in the IP address prefix that are equal (the longer the common prefix, the closer they are). In this case, every client within a certain range of a client, whose available bandwidth is known, would be treated alike.

Determination of the shaping parameters is indicated in 850. It is described in more details in FIG. 9. Once the shaping parameters are determined, they are communicated to the server in 860. The server can react in three ways when it gets the shaping parameter report indicating that client constraints exist. These are indicated with 870:

If no prior shaping rule has been in place a new rule would be set up (using the trigger event 440) that limits the flow (all the connections) between the server and the client. Such a rule would typically look like {source address=server address; source port=*; destination address=client address; destination port=*; application=*;+shaping parameters}.

If a shaping rule is already in place (due to an earlier or ongoing parallel session with the client) this rule might have to be modified using a trigger event 442. The shaping parameters would be adapted according to the new bandwidth constraints.

If the server realizes that the bandwidth constraints have disappeared, (e.g. the client has terminated a voice over IP call with a peer) the rules pertaining to the client can be deleted using a trigger event 442. This allows the server to make full use of the existing bandwidth between itself and the client.

Finally, the server sends the requested data in 880, shaped according to the best matching rule. Note that the FIG. 8 is only an example of one possible message flow. For example, the report of shaping parameters 860 could be repeated periodically or whenever there is a significant change in bandwidth. It is also possible that the server starts sending data right away 880, using a shaping rule of a previous session with the same client.

FIG. 9 is an optional client process 900 that is executed on the client to communicate client constraints to the server. This is necessary for a network server to determine the shaping parameters. The client process is installed permanently or is deployed from the server as described in FIG. 8. It provides a more detailed description of step 850 in FIG. 8. The optional client process 900 impacts the server 170 in so far that the communication of client constraints can trigger the events 440–442 in FIG. 4.

After initialization 905, the client determines the bandwidth (step 910) with which it is connected to the network. This can be done by detecting the modem parameters or by prompting the user for the corresponding input. The client process then determines the bandwidth used by the real time media transmission that is supposed to be protected from ordinary data transmission in step 920. This can be done by analyzing the media software configuration or by prompting the user.

The 'free' bandwidth, i.e. the bandwidth available for other transmission like downloading data from the network server ist then determined in step 930. As an example, if the average bandwidth used by the media transmission is 10 kbit/s and the modem speed is 28 kbit/s the 'free' bandwidth is 18 kbit/s. Usually, packets do not arrive nicely spaced out. Therefore, the free bandwidth should be reduced to compensate for variation in the arrival of packets for both sessions to guarantee a reasonable delay and delay variation for the media transmission.

Steps 920 and 930 can also be determined by analyzing the quality of the media transmission. In this case, the client process keeps track of the average delay of media transmission packets (e.g. by using the statistics provided by the Real Time Control Protocol RTCP). When the delay increases, this signifies that the available bandwidth from the server should be decreased. When the delay decreases, the available bandwidth from the server can possibly be increased. It is even possible to give the full bandwidth to the server in cases where the media transmission is paused temporarily (e.g. pause within a phone call, pause due to rewinding of a video tape on the client/source, etc.).

In step 940, the available bandwidth is reported to the server. The client program can calculate the shaping parameters either directly, or provide the server with the acquired information. Reports might be sent in regular intervals or only when the available bandwidth between the client and the server changes significantly.

In the preceding descriptions shaping rules have been explained in greater detail (e.g. in FIG. 7). Note, that a rule can be used for other purposes as well. The main goal of a rule is to control the behavior of a set of sockets using a bidirectional association. Above, different behavior was related to setting different bandwidth limits to different sets. Actually, different rules pertaining to the same flow can form a rule set.

Other uses of this invention include, but are not limited to, the following.

Rate shaping a flow (a set of network sessions) with the purpose of reducing the probability that subsequent packets of the same sessions get lost due to a buffer overflow in or at the edge of the network. For streaming applications, where video and/or audio is sent while being simultaneously played out at the client side, limiting the bandwidth and pacing out the packets can limit the necessary buffers at the receiving (client) end. In addition, experiments show that rate shaping (pacing

out the packets in constant intervals) reduces the average packet loss rate, the average packet delay, and the average packet delay variations.

A network server might be subject to security considerations. For example, a company WWW server might be accessed from employees within the firewall, but also from employees working remotely, outside the firewall. Setting up rules allows to selectively encrypt, authenticate flows to outside employees to guarantee privacy and authenticity. Instead of defining bandwidth parameters, the shaping parameters would consist of security parameters like the encryption technology, the encryption keys, etc.

Rate shaping a flow (a set of network sessions) with the purpose of differentiating priorities of packets between different flows. This can be useful e.g. to give faster access to customers who order products over a web site, than to visitors of the web site who are just looking for general information. An end system usually has access to information that a router in the network does not possess. Therefore, it is possible to describe a flow based on the application type as specified at the end system. The flow parameters of a rule do not necessarily need to contain all parameters. It is always possible to put a wildcard value in an unknown parameter. Therefore, it is possible to give priority to a secure WWW transaction (listening on port 443) over a regular WWW transaction (listening on port 80) by limiting the bandwidth of all flows using source port 80.

We claim:

1. A server connected through one or more network interfaces to one or more networks, each of the networks connected to one or more clients, the server having one or more memories and one or more central processing units (CPUs) and further comprising:

one or more applications executed by one or more of the CPUs, each application using one or more sockets connected to the networks to communicate over the networks;

one or more rule sets containing one or more rules; and

one or more socket sets of one or more of the connected sockets, each of the socket sets associated with one of the rule sets, the rule set controlling one or more packets sent by the applications on each of the sockets in the associated socket set;

wherein the rule set controls the timing of the sending procedure of the packets to the network or the sending of received packets to the application on any socket in the associated socket set in one or more of the following ways: limiting a peak rate of delivered packets, limiting size of a burst of packets delivered at the peak rate, and limiting size of sent packets.

2. A server, as in claim 1, where the association between a rule set and a socket is based on any one or more of the following attributes of the socket: a source address of the server, a destination address of the client, a source port of the server, a destination port of the client, a protocol type, and an application type.

3. A server, as in claim 1, where the rule set controls the security of the packets sent on any of the sockets in the associated socket set in one or more of the following ways: using an encryption key, using the encryption technique, how often to encrypt, packet authentication, how often to authenticate packets, and method of authentication, when to prevent packet sending.

4. A server, as in claim 1, where the rule set controls all the packets sent from any socket in the associated socket set in the same way.

5. A server, as in claim 1, where the association between one or more of the socket sets and the rule set is bidirectional.

6. A server, as in claim 1, where the rule set controls the temporal spacing of successive packets to slow down the transmission.

7. A server, as in claim 6, where one of the applications is one of the following: a video stream, an audio stream, a video and audio stream, and a large data stream.

8. A method, executed by a computer system, the computer system having one or more network connections to one or more networks, the method comprising the steps of:

verifying that there is information in the form of packets to be communicated between one or more applications executing on the computer system over one or more of the network connections;

verifying that the information can be communicated as determined by one or more rules in a rule set associated with one of the network connections, wherein the rule set is accessible for associating with one or more of the network connections;

communicating the information if the rules in the rule set are satisfied; and

wherein rule set controls the timing of a sending procedure of the packets to the network or the sending of received packets to an application on any socket in an associated socket set in one or more of the following ways: limiting a peak rate of delivered packets, limiting size of a burst of packets delivered at the peak rate, and limiting size of sent packets.

9. A method, as in claim 8, where the information is modified to modified information if the rules are not satisfied, the modified information satisfying the rules.

10. A computer connected through a connection on one or more network interfaces to one or more networks, the computer comprising:

means for verifying that there is information in the form of packets to be communicated between one or more applications executing on the computer system over one or more of the network connections;

means for verifying that the information can be communicated as determined by one or more rules in a rule set associated with one of the network connections, wherein the rule set is accessible for associating with one or more of the network connections;

means for communicating the information if the rules in the rule set are satisfied; and

wherein the rule set controls timing of a sending procedure of the packets to the network or the sending of received packets to an application on any socket in an associated socket set in one or more of the following ways: limiting a peak rate of delivered packets, limiting size of a burst of packets delivered at the peak rate, and limiting size of sent packets.

11. A server connected through one or more network interfaces to one or more networks, each of the networks connected to one or more clients, the server having one or more memories and one or more central processing units (CPUs) and further comprising:

one or more applications executed by one or more of the CPUs, each application using one or more sockets to communicate over the network;

one or more rule sets of one or more rules, one or more of the rules containing one or more client constraints; and

one or more socket sets of one or more of the connected sockets, each of the socket sets associated with only one of the rule sets, the rule set controlling one or more packets sent by each of the sockets in the associated socket set.

12. A server, as in claim 11, where the client constraints include any one or more of the following: a client with limited access bandwidth to the network, a client with limited bandwidth to the server, one or more other network sessions involving the client, a client with limited computational power, and a client with limited memory.

13. A server, as in claim 11, where the client constraints are communicated to the server from a client constraint determinator that executes on the client to determine the client constraints.

14. A server, as in claim 11, where the client constraints are communicated to the server from an agent computer that has a concurrent second session with the client.

15. A server, as in claim 11, where the client constraints are assumed or extrapolated based on previous observations, or communication with clients with similar constraints, or clients located close to each other.

16. A server, as in claim 11, where the client constraints are determined using human input.

17. A server, as in claim 11, where client constraint is a client bandwidth constraint and the rule set controls the packets to be sent slower because the client has concurrent audio and/or video sessions.

18. A server, as in claim 17, where the rule set controls using any one or more of the following shaping mechanisms: limiting the average rate of sent packets, limiting the peak rate of packets, limiting the size of the burst of packets that are sent out at the peak rate, and limiting the size of the packets sent.

19. A server, as in claim 11, where the client constraints are thin client constraints.

20. A server, as in claim 19, where the thin client constraints are imposed by any one or more of the following thin clients, that are tier-zero devices: a set-top box, a hand held device, a Palm Pilot, a Web appliance, and a Web-based application device.

21. A server connected through one or more network interfaces to one or more networks, each of the networks connected to one or more clients, the server having one or more memories and one or more central processing units (CPUs) and further comprising:

one or more applications executed by one or more of the CPUs, each application using one or more sockets connected to the networks to communicate over the networks;

one or more rule sets containing one or more rules, wherein the rule sets are data structures for associating with one or more sockets;

one or more socket sets of one or more of the connected sockets, each of the socket sets associated with one of the rule sets, the rule set controlling one or more packets sent by the applications on each of the sockets in the associated socket set; and

wherein the rule set controls the timing of the sending procedure of the packets to the network or the sending of received packets to the application on any socket in the associated socket set in one or more of the following ways: limiting a peak rate of delivered packets, limiting size of a burst of packets delivered at the peak rate, and limiting size of sent packets.

* * * * *